設計モデル検証(応用編)
―第一回:並行分散システムの設計の難しさ―

2009年9月7日
NII 吉岡信和

この講義のゴール

- 何を設計・検証しているのかを把握できる
  - どういう前提・条件・範囲で設計しているのか？
  - その前提・条件・範囲のもとでの意味のある検証になっているのか？
- 前提・条件・範囲を明確しつつ設計、検証ができる
  - 適切な検証ツールの使い分けができる
  - さまざまな検証ノウハウを活用できる

➔ 並行分散システムの本質を捉え、明確化し、正しさを証明できる人材

今日の講義内容

- 基礎編で何を学んだか?
  - 分散システムの難しさ
  - 検証プロセスの概要
- 応用編では何を学ぶか?
  - システムの制約を考慮した設計と検証
  - 環境のモデリング
- 並行システムのモデル化法(演習)
- 外部環境を考慮した設計モデルの構築(演習)

基礎編の復習＋α

検証とは?(復習)

- 真偽を確かめること。事実を確認・証明すること。(大辞林第二版)
  - ソフトウェアの検証:ソフトウェアに関する事実の確認・証明
- 英語ではverification
  - Software verification is a broad and complex discipline of software engineering whose goal is to assure that a software fully satisfies all the expected requirements.
    (Wikipedia英語版)
    「ソフトウェアが、全ての期待される要求を完全に満たすことの確認」

検証とは?(復習)

- 2種類の検証:動的・静的
  - 動的検証:ソフトウェアを動作させて検証
    - 例:テスト
  - 静的検証:ソフトウェアを動作させずに検証
    - 例:プログラム解析、形式的検証
- モデル検査は、形式的検証手法の1つ
  - 形式的検証:数学的手段により、形式的に記述されたソフトウェアの仕様を厳密に検証すること
  - モデル検査:ソフトウェアの振舞いの仕様に対し、可能な実行パターンを網羅的かつ自動的に検査することにより検証を行う手法

モデル検査とテストとの違い（復習）

項目	モデル検査	テスト
動的・静的	静的	動的
振舞いの網羅性	網羅的	部分的
検証の厳密性	厳密	不確実
利用容易性	難	易
検証可能なソフトウェアの規模	小規模	大規模
利用実績	少	多

青字は優位な項目を示す

なぜ検証が必要か?
ーシステムの高度分散化ー

【分散化】:従来単一システムにまとめられていた個別の機能としてノード化され、ネットワークで接続・連携

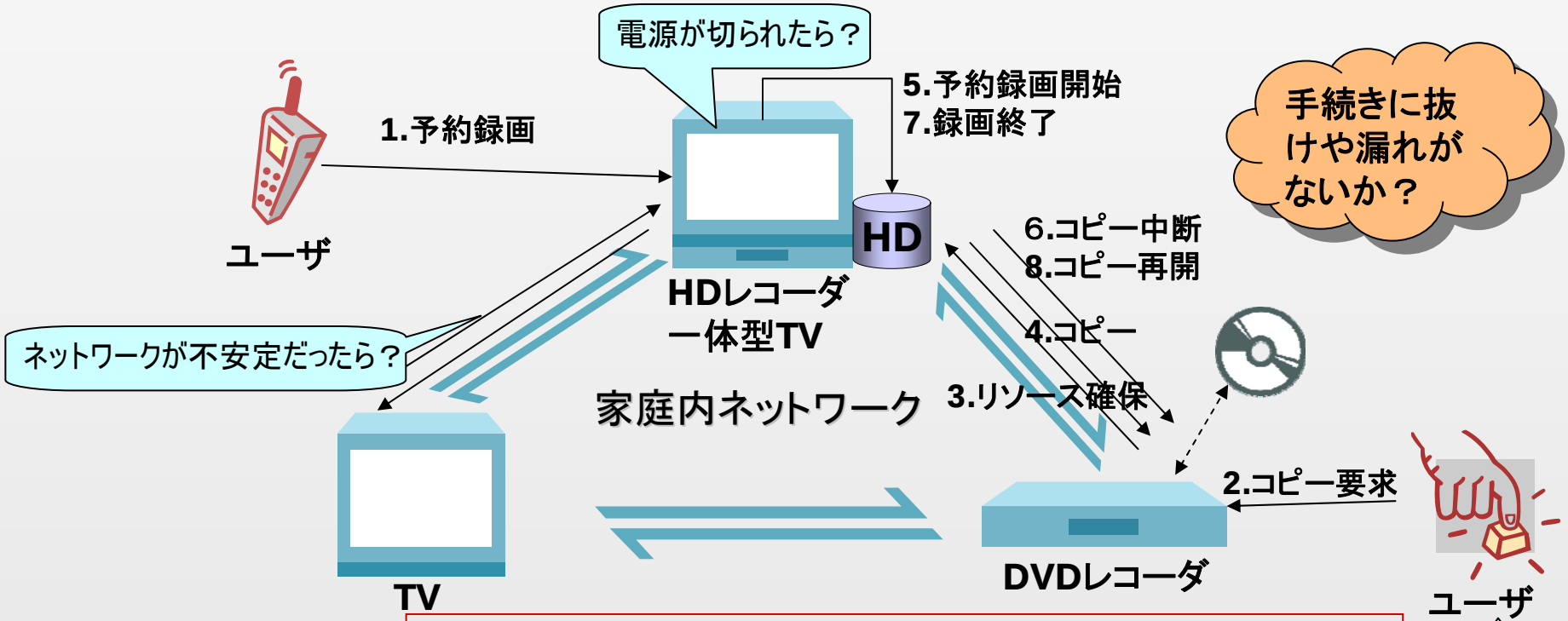
➔振る舞いの可能性が組み合わせ的に増大

【家電製品、ネットワーク接続、オープン化】

➔さまざまな例外を考慮して設計する必要がある

分散システムにおける検証

- 多種多様な**外部環境を考慮**しないといけない
- **様々な動作パターン**に対し、人海戦術によるテストはもはや限界

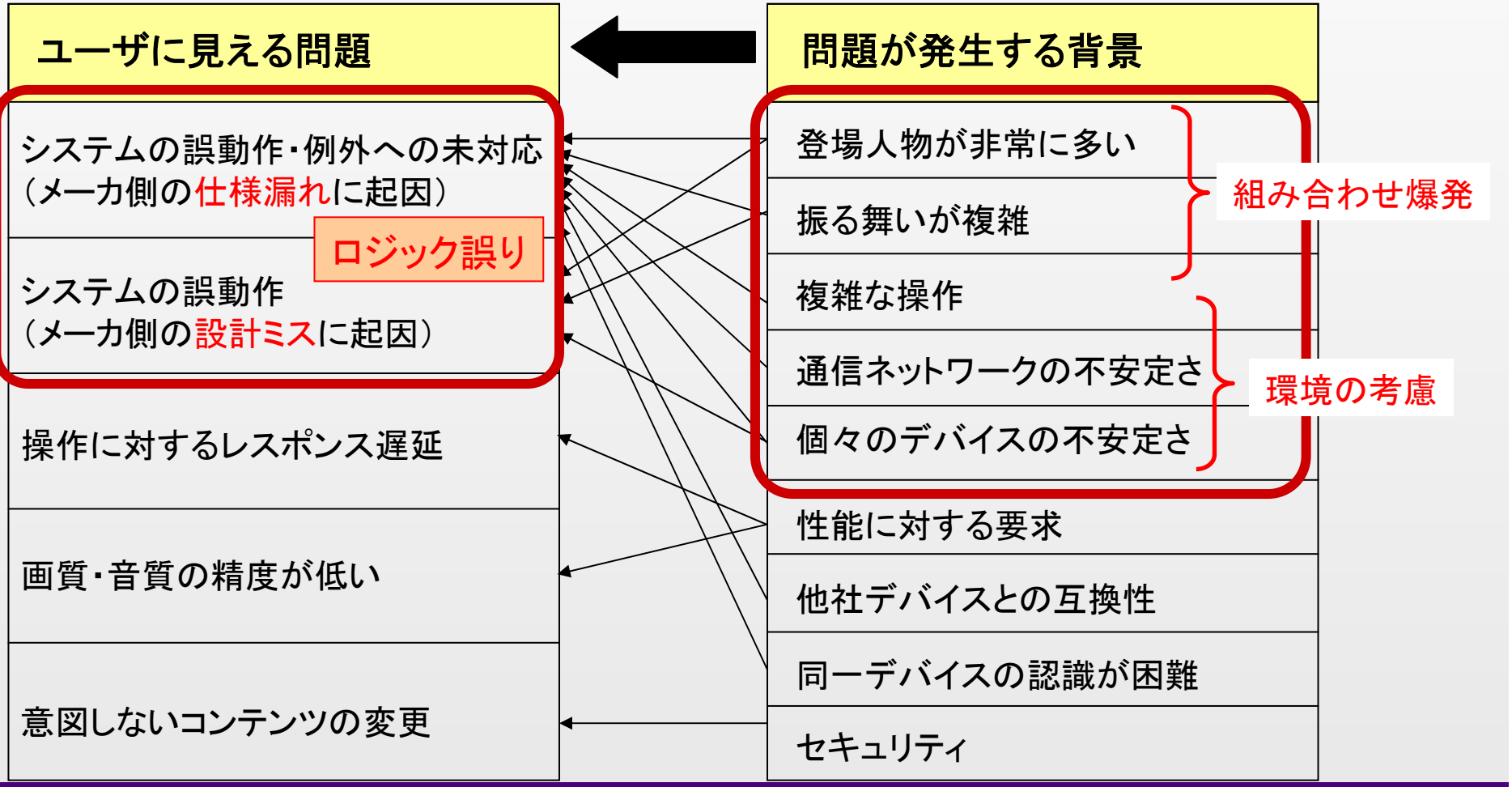


テストの限界

項目	モデル検査	テスト
動的・静的	静的	動的
振舞いの網羅性	網羅的	部分的
検証の厳密性	厳密	不確実
利用容易性	難	易
検証可能なソフトウェアの規模	小規模	大規模
利用実績	少	多

この部分の短所が致命的になりつつある

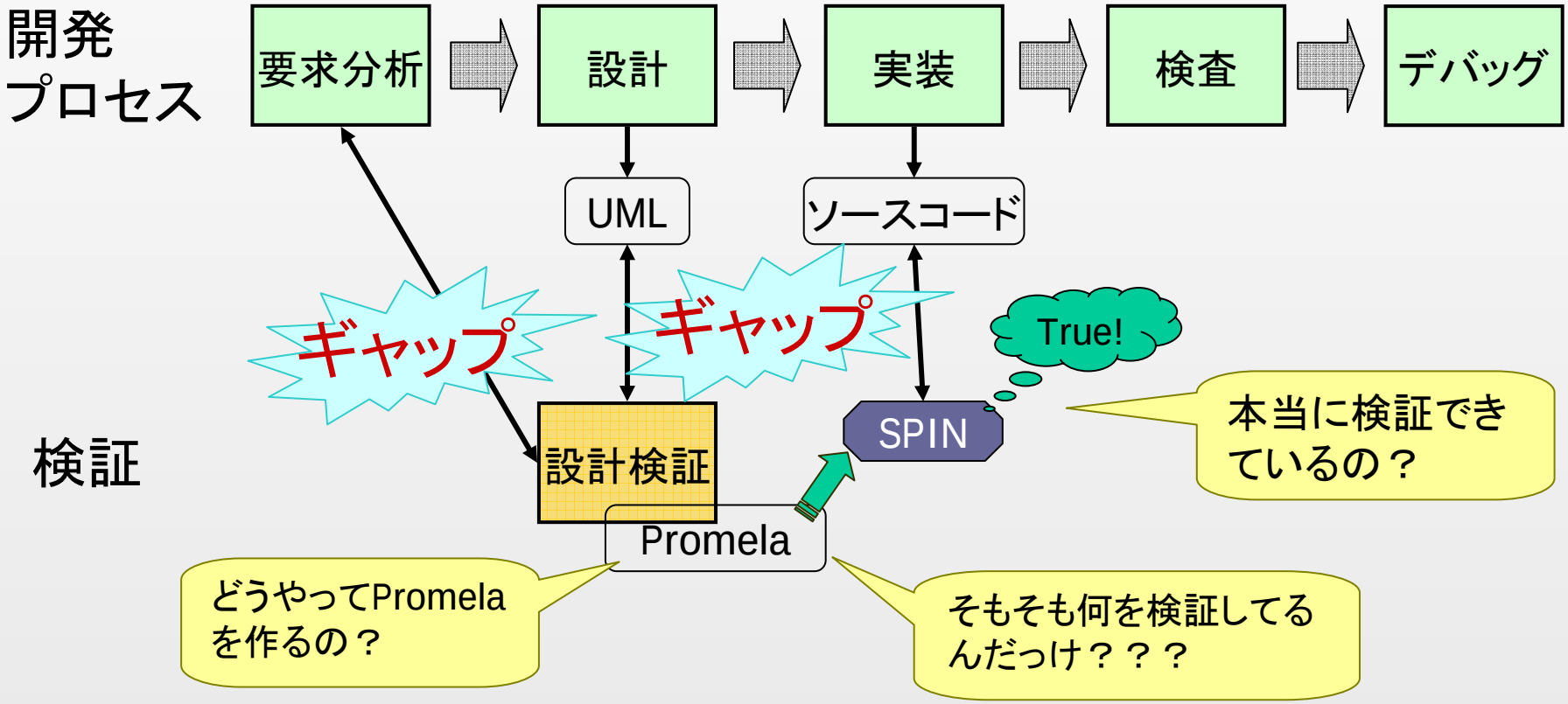
モデル検査が解決してくれる問題



設計モデルの検証の難しさ

- 検証したいモデル（設計モデル）と検証のための記述（検証モデル）の間に大きいギャップ
- 何を検証したいかにより、システムの外部環境やシステムの例外をどこまで考慮し、モデル化するかが異なる
- 何を設計したいかにより、並行システムをどう同期・非同期でモデル化するが異なり、検証方法も異なる

従来の設計検証



検証の難しさ

- 設計と検証のモデル変換と等価性の確認
- 最適化：状態爆発の回避、問題領域の限定
  - 抽象化
  - 関連部分の切り出し、効率化のための変換
- 完全なモデルの構築
  - システム外部の振る舞い（環境）のモデル化

    例）ノードの電源断、リセット、ユーザの振る舞い、ネットワークの不安定さ
  - 並行動作・通信のモデル化
    - 同期なのか？非同期なのか？

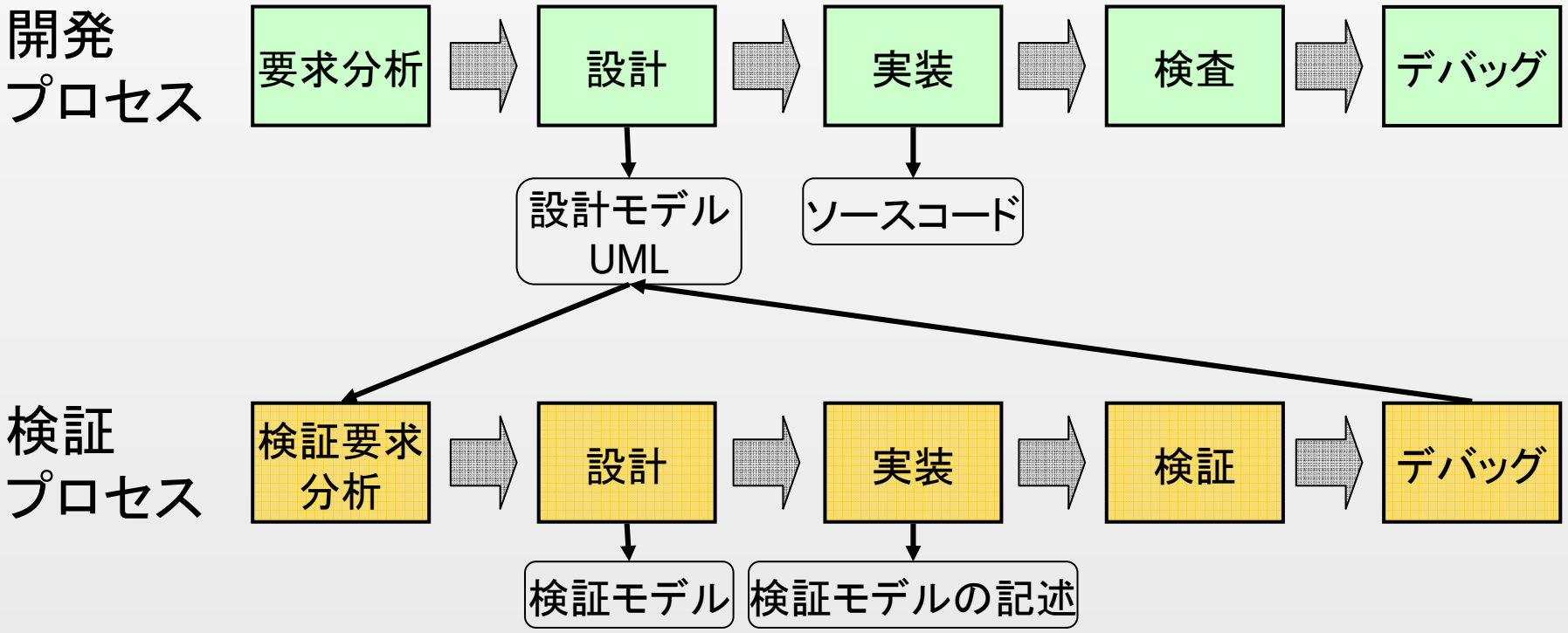
検証の難しさの例

- 設計と検証のモデル変換と等価性の確認
  - → 正しく状態が遷移しているの？
  - → 正しくイベントが受け取れているの？
- 最適化：状態爆発の回避、問題領域の限定
  - 抽象化
    - → どの状態・イベントを区別すればよいのか？
  - 関連部分の切り出し、効率化のための変換
    - ➔検証したい状態・イベントはどこなのか？
- 完全なモデルの構築
  - システム外部の振る舞い（環境）のモデル化
    - → 人はどういう振る舞いをするのか？
    - → 通信は安定なのか？
  - 並行動作・通信のモデル化
    - → イベントはどういうタイミングで相手に伝わって消えるのか？
    - → どういうタイミングで遷移するのか？

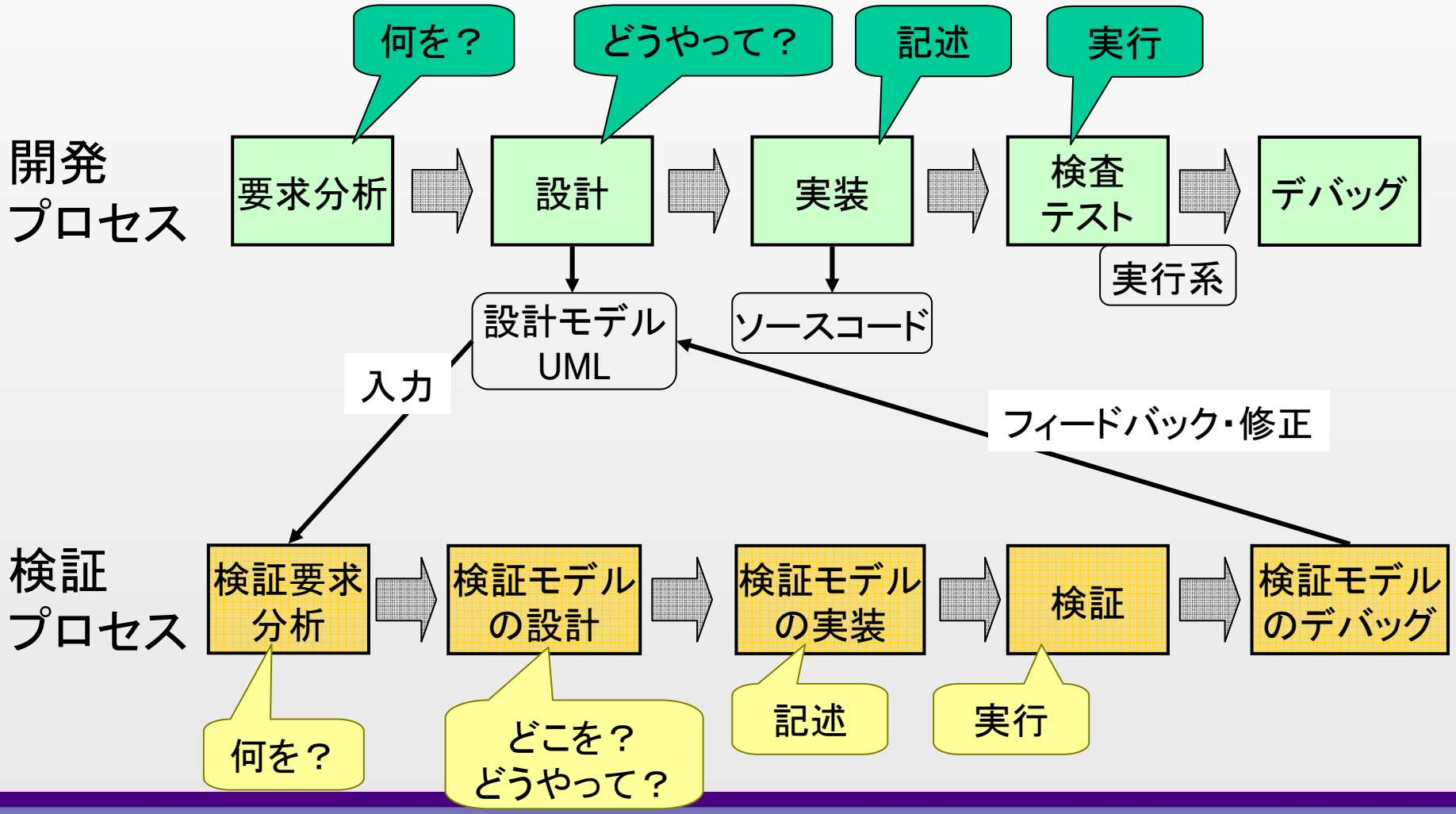
応用編で扱う技術

- 設計と検証のモデル変換と等価性の確認
  - ➔設計、検証したいモデルの特徴から検証ツールの使い分け
- 最適化：状態爆発の回避、問題領域の限定
  - 抽象化
  - 関連部分の切り出し、効率化のための変換
  - ➔検証のためのノウハウ
- 完全なモデルの構築
  - システム外部の振る舞い（環境）のモデル化
    例）ノードの電源断、リセット、ユーザの振る舞い、ネットワークの不安定さ
  - ➔例外を考えた環境のモデル化法
  - 並行動作・通信のモデル化
    - 同期・非同期
  - ➔設計したい粒度・システム制約に合わせたシステムの設計法

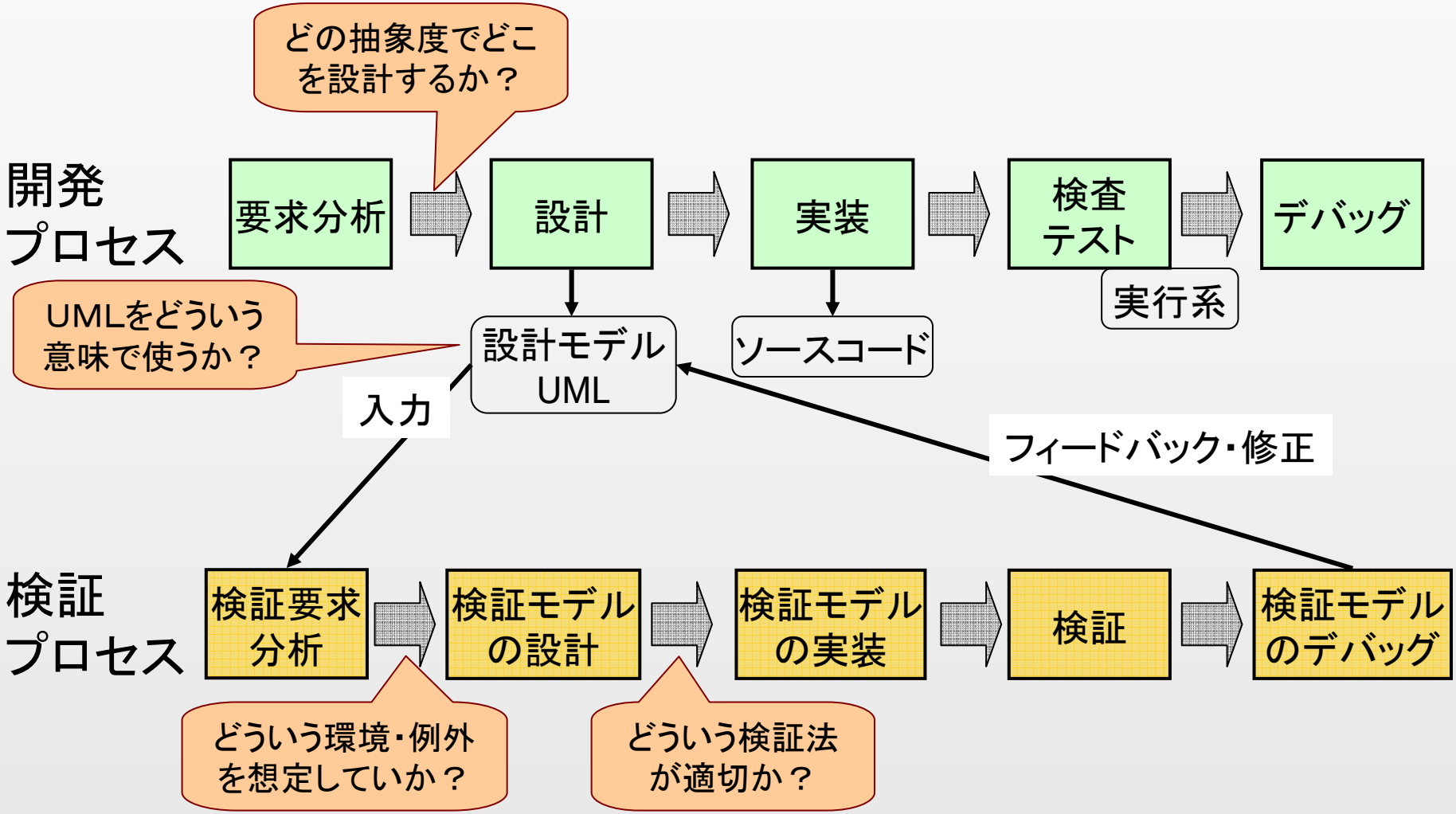
検証のためのプロセス



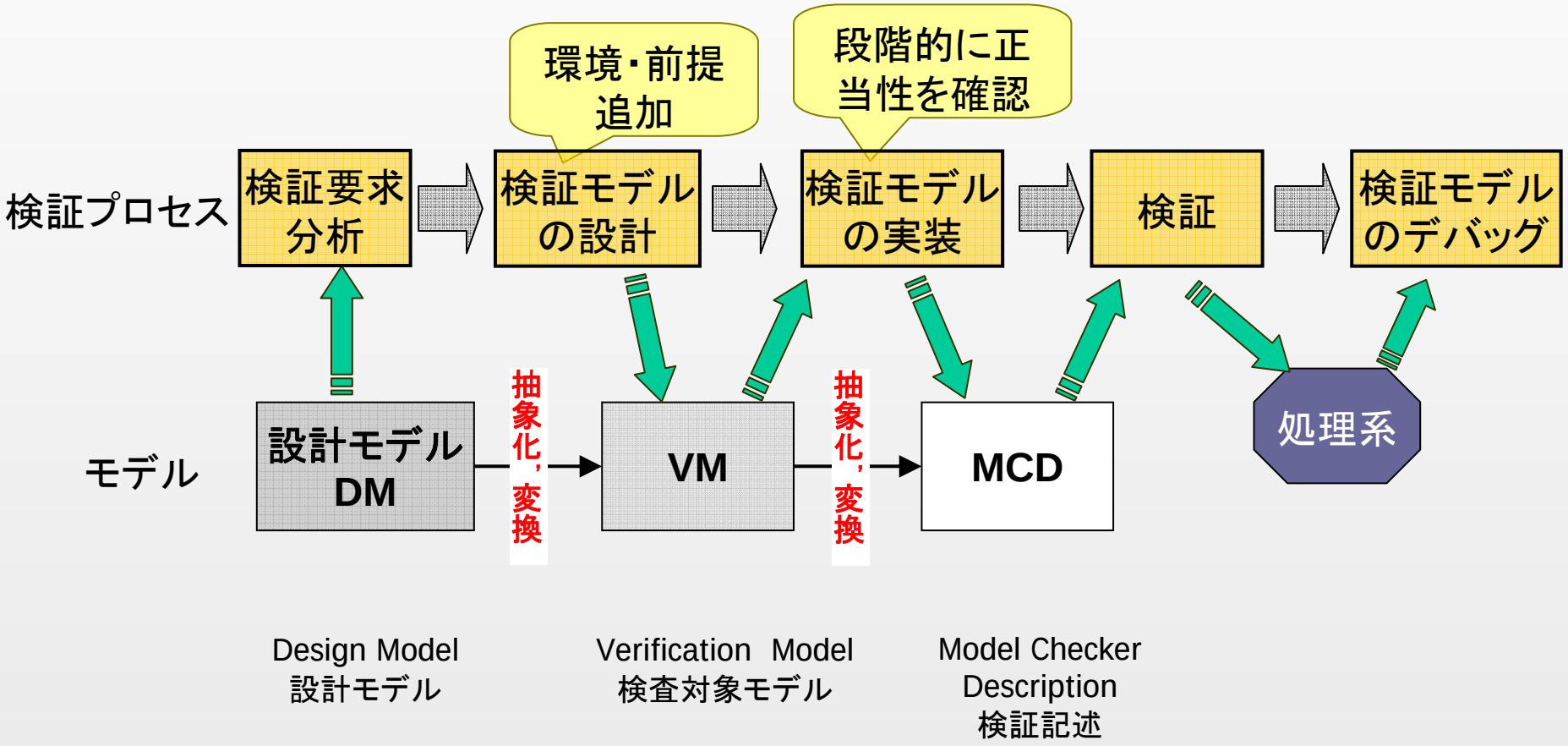
検証プロセス



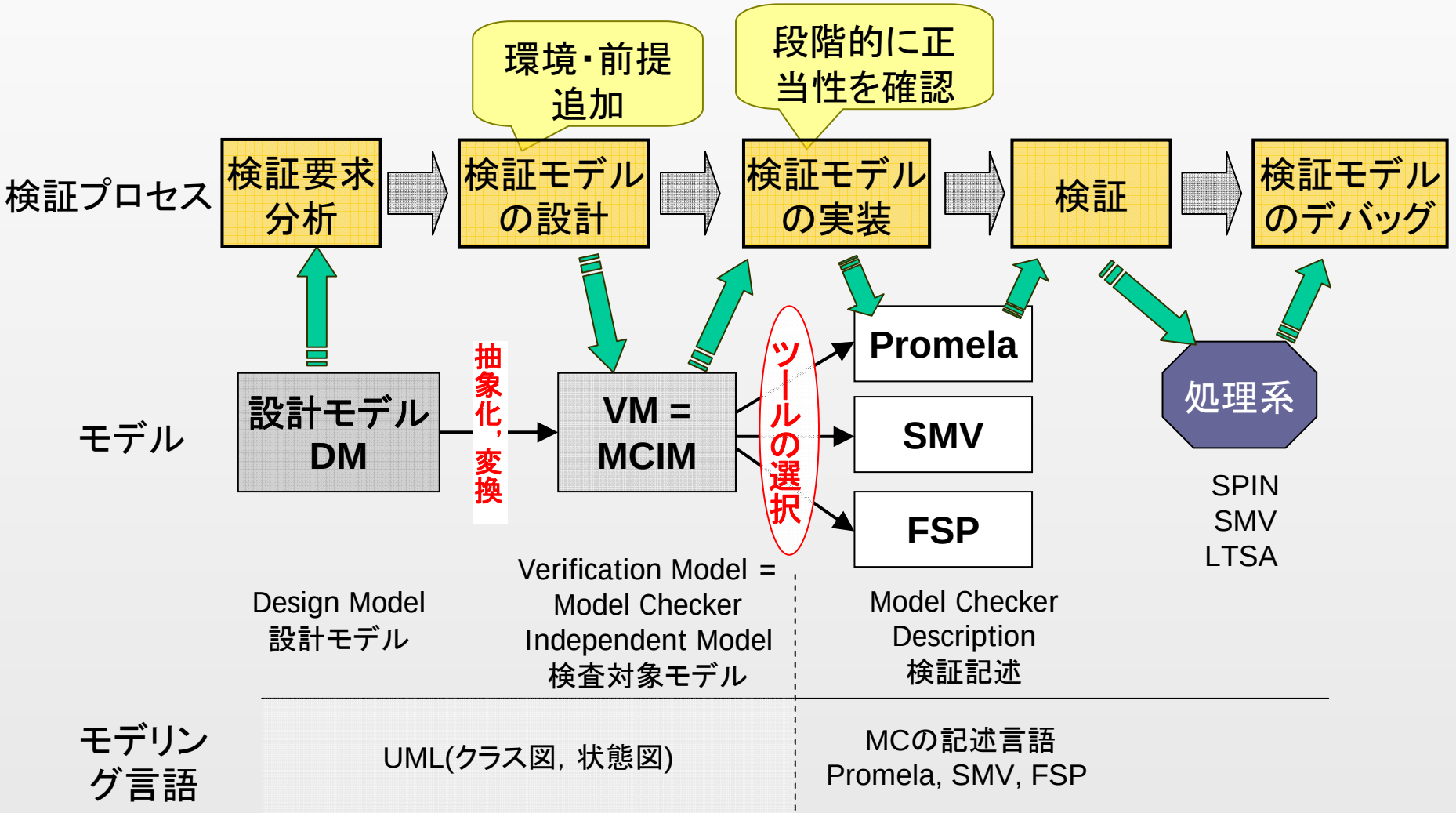
検証プロセス：応用編で扱うこと



検証のためのモデル

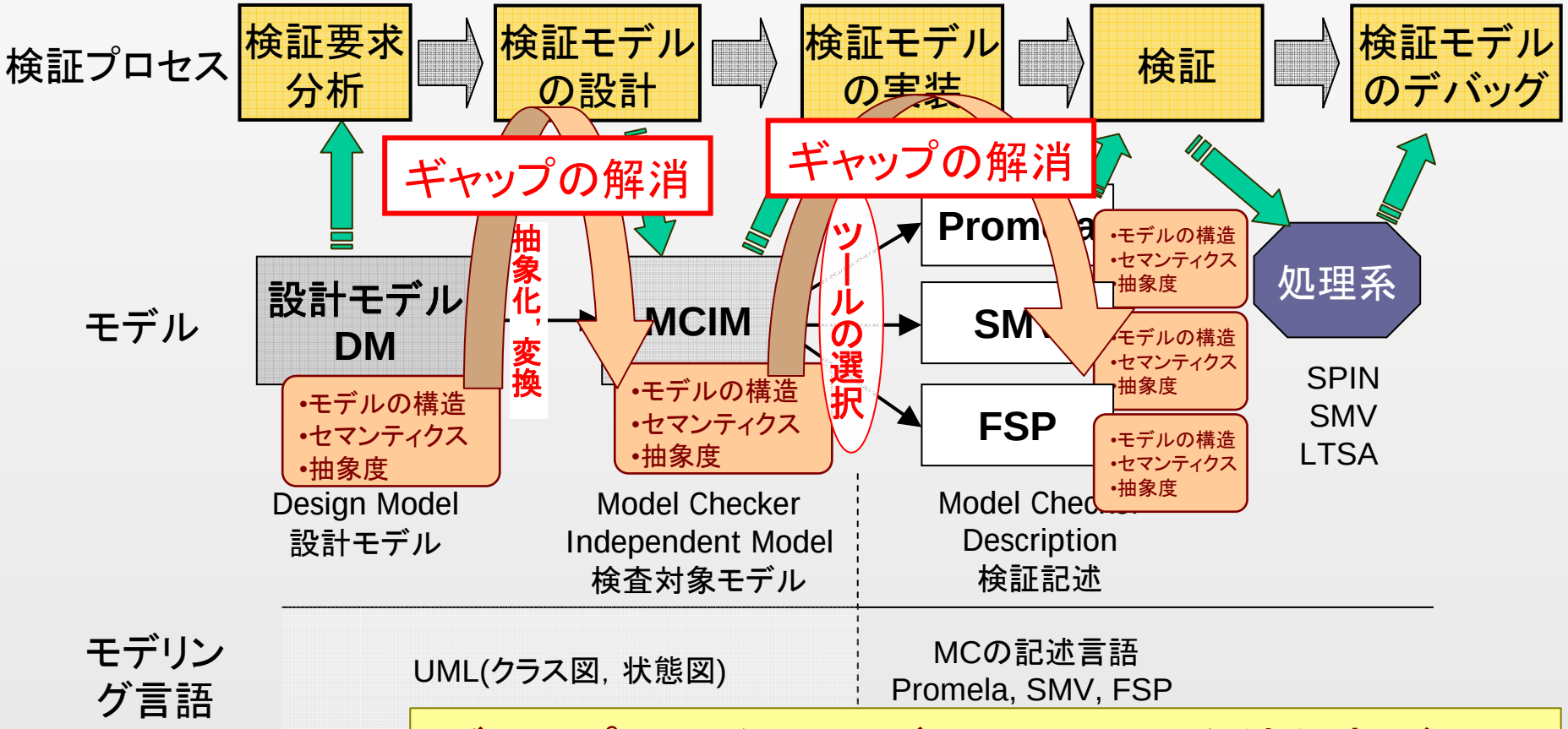


検証のためのモデル：ツールの使い分け



モデル間のギャップの解消

検証は、設計モデルと検証モデルとの間のギャップを検証したい部分の意味を崩さずに段階的に解決するプロセス



ギャップの少ないモデルやツールを使う事が重要

応用編で扱う設計・検証ノウハウ

1. 抽象度・システム制約を考慮した設計モデルの構築
   - 同期・非同期モデルによる設計と検証
2. 外部環境を考慮した検証
   - 例外を考慮と環境の振る舞いの設計と検証
3. システム制約、検証したい性質に合わせた検証ツールの使い分け
   - SPIN、SMV、LTSA
4. 検証ノウハウ

応用編の内容

本日
1. 抽象度・システム制約を考慮した設計モデルの構築
   - 同期・非同期モデルによる設計と検証
2. 外部環境を考慮した検証
   - 例外を考慮と環境の振る舞いの設計と検証

8コマ＋演習5コマ
3. システム制約、検証したい性質に合わせた検証ツールの使い分け
   - SPIN、SMV、LTSA

1コマ
4. 検証ノウハウ

応用編のシラバス(詳細)

- 第1回:導入、基礎編復習、並行分散システムの設計の難しさ
- 第2回:SPINにおける分散システムの難しさの検証(1) – 同期・非同期モデルと検証
- 第3回:SPINにおける分散システムの難しさの検証(2) – 環境モデルと検証
- 第4回:SMV概論 – モデル記述、検証、反例分析、シミュレーション、設計モデル修正
- 第5回:SMVにおける分散システムの難しさの検証(1) – 同期・非同期モデルと検証
- 第6回:SMVにおける分散システムの難しさの検証(2) – 環境モデルと検証
- 第7回:LTSA概論– モデル記述、検証、反例分析、シミュレーション、設計モデル修正
- 第8回:LTSAにおける分散システムの難しさの検証(1) – 同期・非同期モデルと検証
- 第9回:LTSAにおける分散システムの難しさの検証(2) – 環境モデルと検証
- 第10回:検証のためのノウハウ詳論
- 第11回:演習(1) – 応用課題:設計モデルの特徴とツールの使い分け
- 第12回:演習(2)
- 第13回:演習(3)
- 第14回:演習(4)
- 第15回:議論とまとめ

抽象度・システム制約を考慮した設計モデルの構築

同期・非同期モデルによる設計と検証

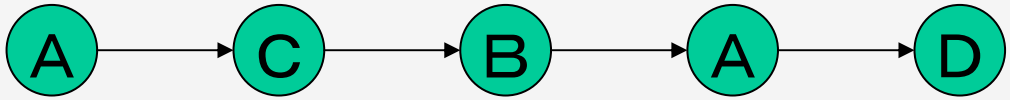
並行分散システムの設計

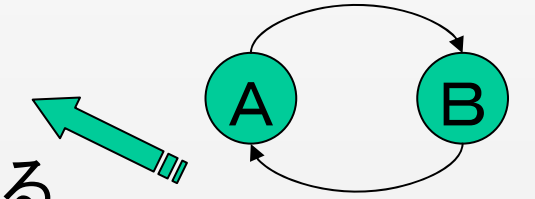
- 設計したいシステムの抽象度やシステムの制約によりモデル化の方法を適切に選択する必要がある
  - 例：システムがクロック同期しながら並列動作
    - バスでつながった組み込み機器、論理回路
    - ロボット制御
  - 例：動作フェーズ・モードが分かれた協調動作で詳細な協調手順を意識しない設計
    - トランザクション処理、オークション、ワークフローなど、通信前後に何をするかに注目した設計

同期・非同期による並行分散システムの設計

【並行動作に関する制約・前提】

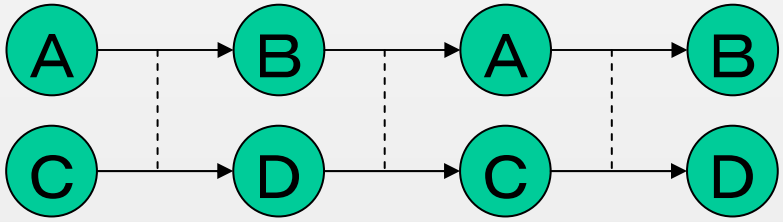
- 非同期遷移：各ノードの遷移順序は非決定的

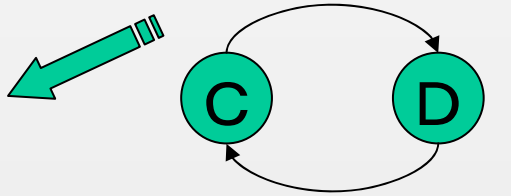




- 同期遷移：同時に全ノードの遷移が起こる

同期並行システム





【通信に対する制約・前提】

- 非同期通信：通信時でブロックしない
  - 実装例）ソケット通信
- 同期通信：実際の通信が起こるまでブロック
  - 実装例）RPC

設計対象による同期・非同期の使い分け

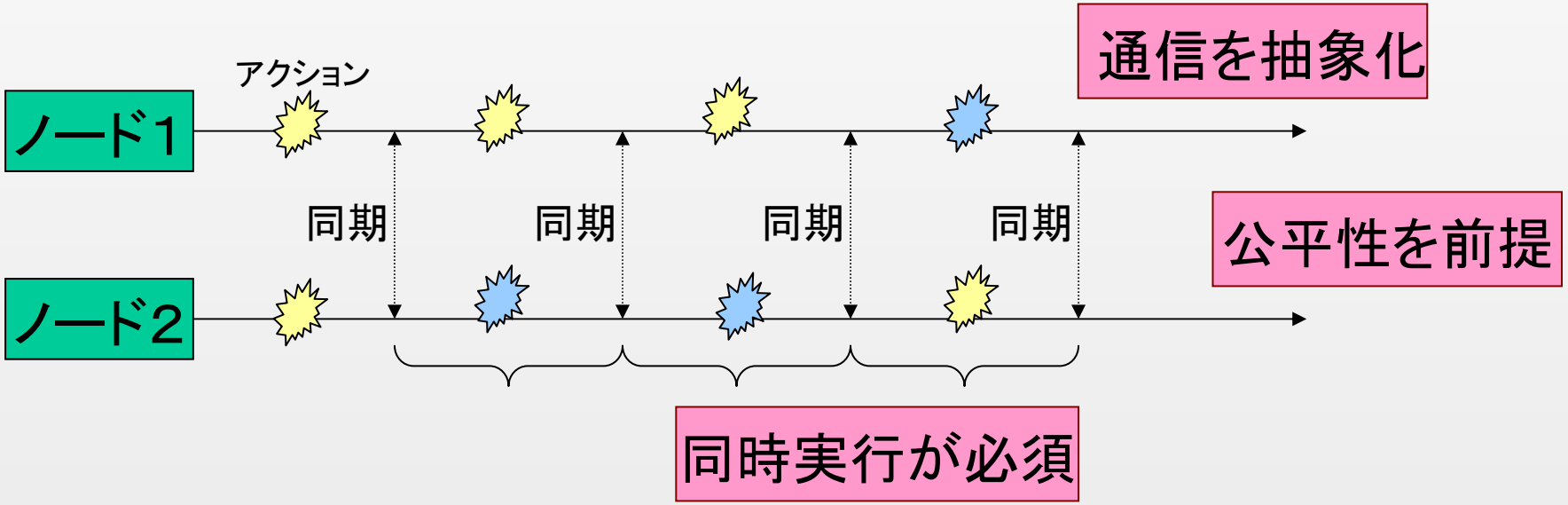
【並行動作に関する制約・前提】
- 非同期遷移：各ノードの遷移順序は非決定的
  - ➔ 個々の動作順・非公平性などを考慮した詳細設計
- 同期遷移：同時に全ノードの遷移が起こる
  - ➔ 前提：並行動作に関して抽象化した設計
    - 公平性を前提（設計の対象でない）
  - ➔ 制約：並行動作に関するシステム制約
    - 動作モデルが決まっている：クロック同期
    - 同期のための動作が個々の動作よりも相対的に長時間かかる
      - 同期メカニズムと個々の動作を別々に設計

同期 ⟺ 非同期
抽象 ⟺ 詳細
制約大 ⟺ 制約小

【通信に対する制約・前提】
- 非同期通信
  - ➔ 具体的な通信手順の詳細設計、プロトコル設計
  - ➔ 並列実行の効率を考えた設計
- 同期通信：実際の通信が起こるまでブロック
  - ➔ 前提：通信の詳細を抽象化した設計、メッセージのタイミングのみに注目
  - ➔ 制約：システムが同期通信しかサポートしていない

同期遷移(同期並行システム)を想定した設計

- 並行動作に関して抽象化した設計
- 並行動作に関してシステムの制約がある



個々のノードのアクションにかかる時間が短く、どういう手順で協調するか(プロトコル)よりも各ノードが振る舞うタイミングを設計

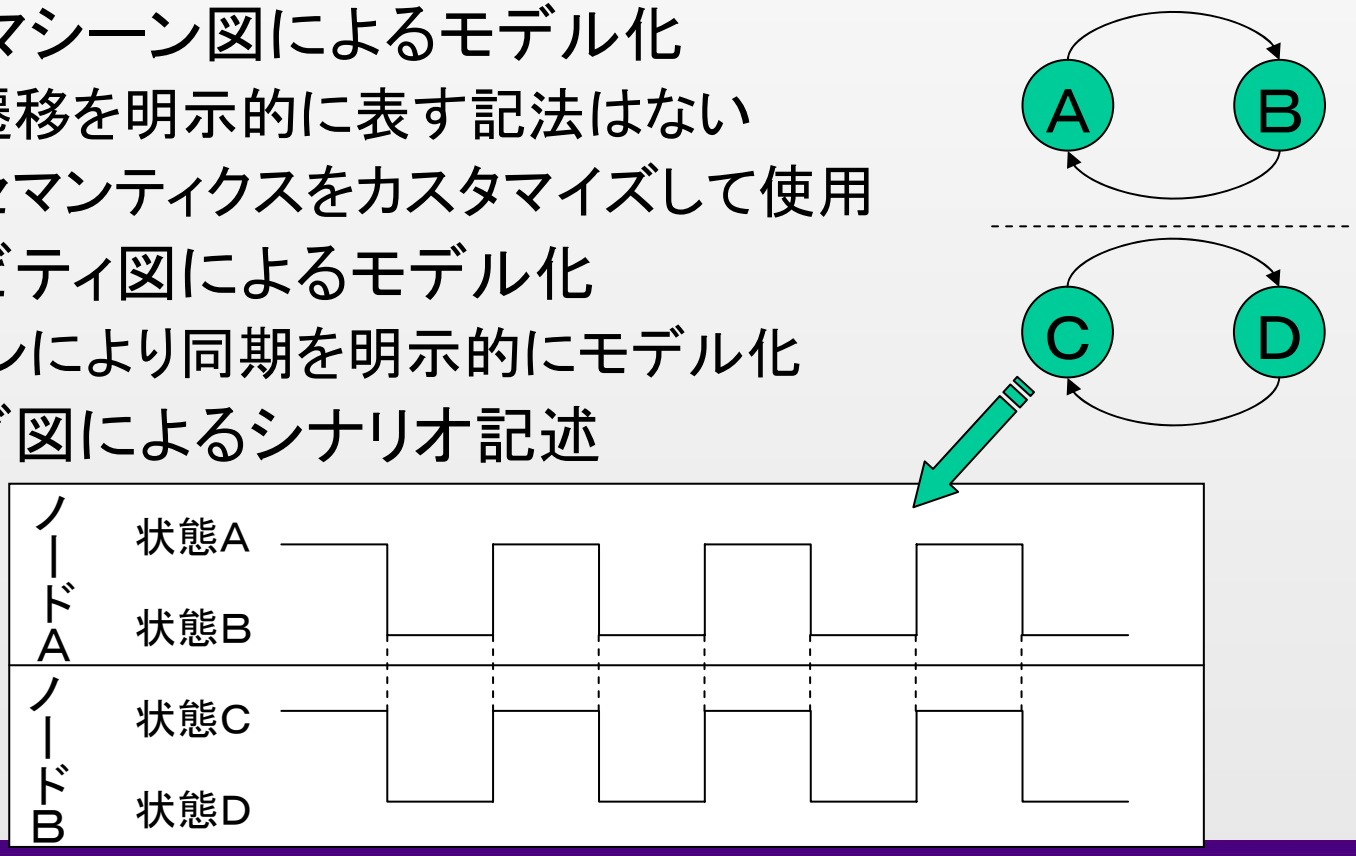
UMLにおける同期・非同期遷移のモデル化

【同時並行システムのモデル化】

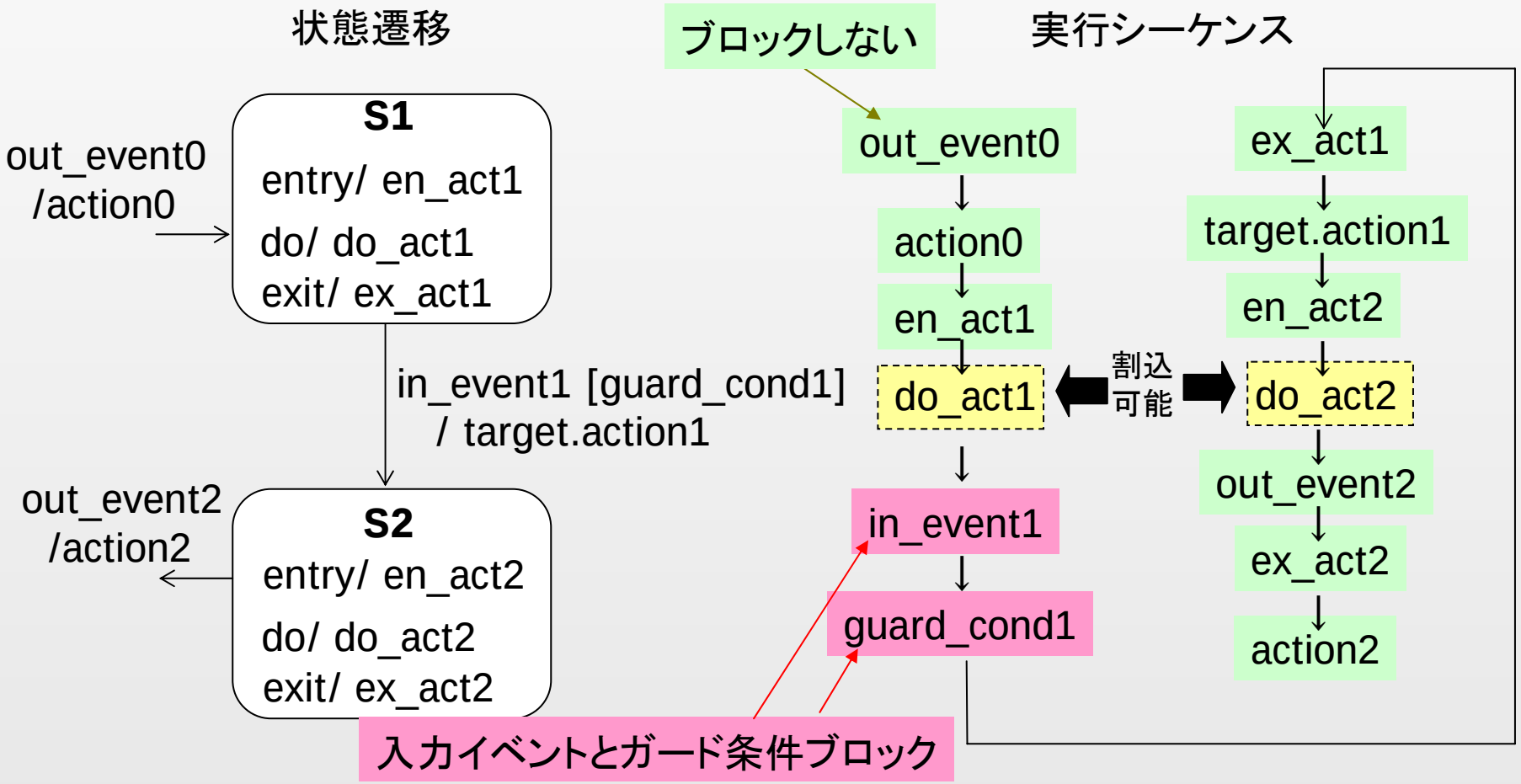
- ステートマシーン図によるモデル化
  - 同時遷移を明示的に表す記法はない

  ➔動作セマンティクスをカスタマイズして使用
- アクティビティ図によるモデル化

  ➔ジョインにより同期を明示的にモデル化
- タイミング図によるシナリオ記述



基礎編で用いた状態マシン図のセマンティクス：非同期遷移、非同期通信（UML標準）

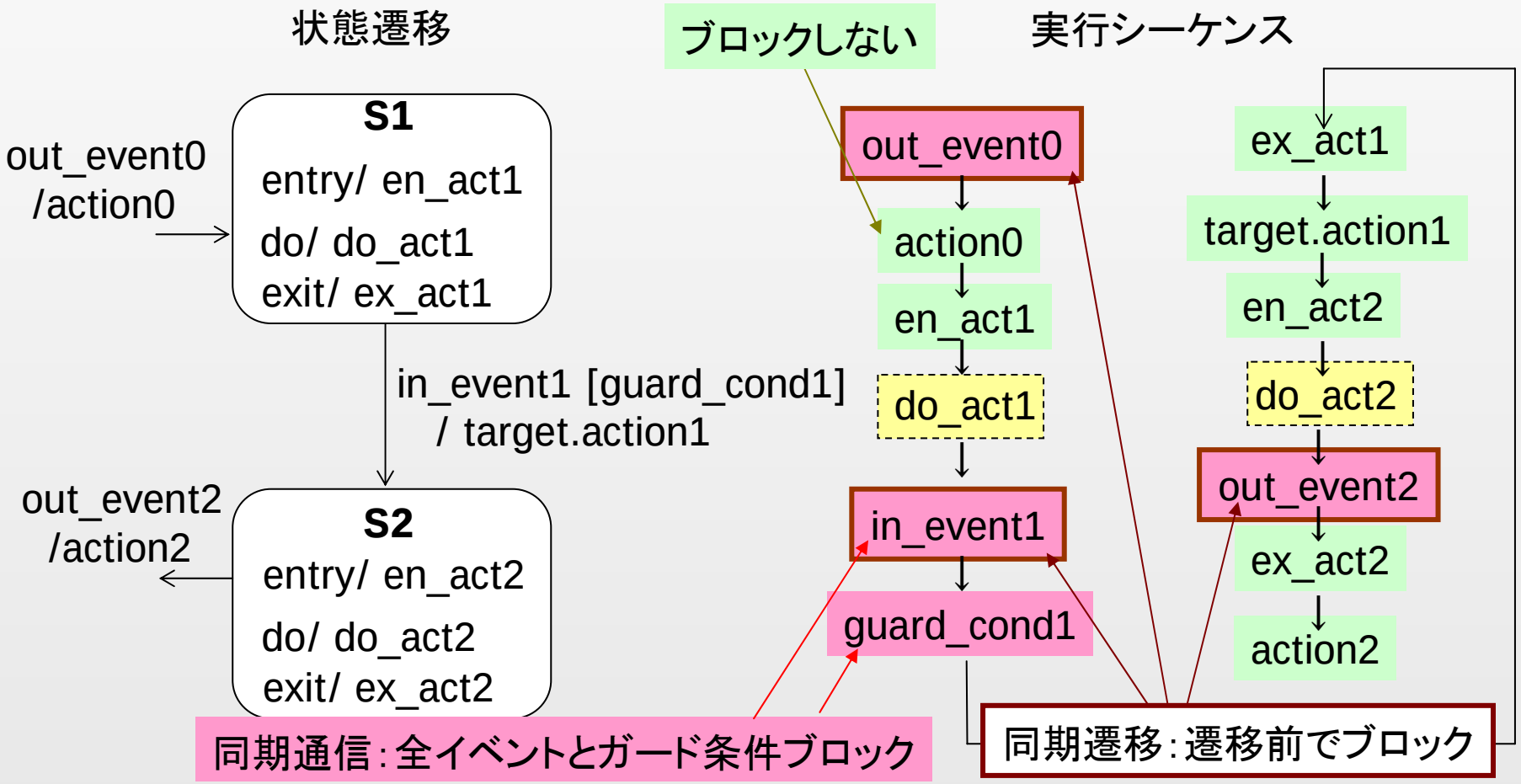


状態マシン図のセマンティクス詳細（UML標準）

- 状態マシン図は、オブジェクト（クラスのインスタンス）における、状態と状態遷移の関係を表す。

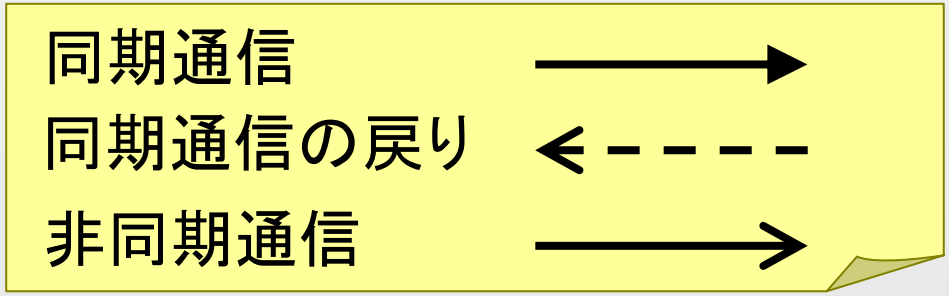
- 注意：
  - 状態間の線上に書かれたe/t.aと[g]は、以下を表す：
    - e: そのオブジェクトが受信するイベントを表す。
    - a: そのオブジェクトが送信するイベントを表す。
    - t: そのオブジェクトが送信するイベントの送信先のオブジェクトが所属するクラスを表す。Self は自分自身のオブジェクトを表す。
    - [g]: ガード条件を表す。gはブール式であり、その式がtrueになった時に遷移が有効となる。ガード条件は、イベントの代わりに利用されることもある。
  - 状態の角丸長方形の中に書かれたentry/, exit/, do/は、以下を表す：
    - entry/ : 何らかのイベントを受信してその状態に遷移した時に、最初に実行されるアクション。
    - exit/ :何らかのイベントを受信してその状態から遷移する時に、最初に実行されるアクション。
    - do / : その状態で実行されるアクティビティ。

同期モデルのセマンティクス例（UML非標準）



UMLにおける同期・非同期通信のモデル化

- ステートマシーン図によるモデル化
  - 同期通信を明示的に表す記法はない：全てのイベントは非同期
  - ➔ステレオタイプで同期通信を明示
  - ➔動作セマンティクスをカスタマイズして使用
- シーケンス図、コラボレーション図によるモデル化
  - ➔矢印の形で区別



演習１：食事をする哲学者のモデル化

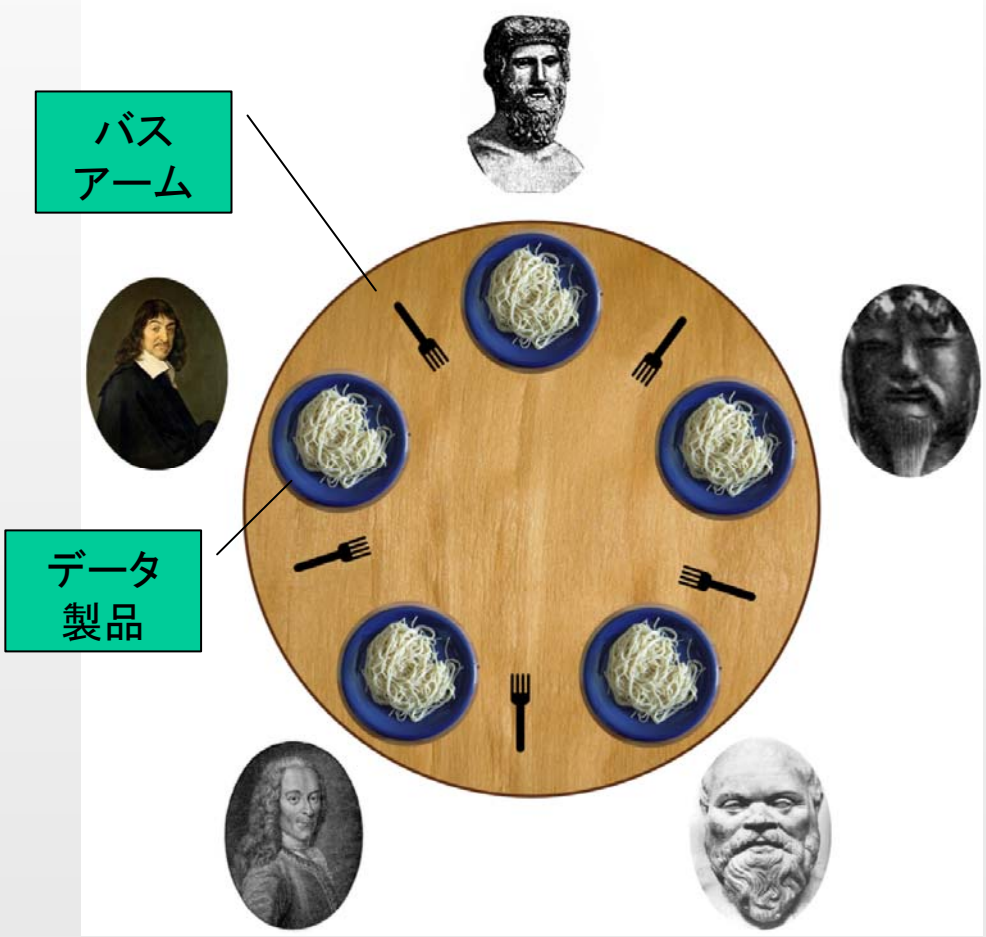
（演習１－Ａ）食事をする哲学者の問題を非同期遷移（非同期並行システム）と同期遷移（同期並行システム）の両方でモデル化せよ

（演習１－Ｂ）各設計モデルが異なることを確認し、それぞれの設計モデルの着眼点、特徴をまとめよ

➔システムの前提・制約により設計すべきモデルが異なる

（演習１－Ｃ）シナリオ・テストに基づき、それぞれのモデルの正当性を確認せよ

食事する哲学者の問題



- 5人の哲学者
  - それぞれ実行単位を表す
- 最初は思考にふけっているが、空腹になるとスパゲッティを自分の皿に盛って食べる
  - ただし、皿に盛るためには両隣のフォークが2本とも必要
  - フォークは共有資源を表す
- 満腹になれば、両方のフォークを置いて再び思考にふける

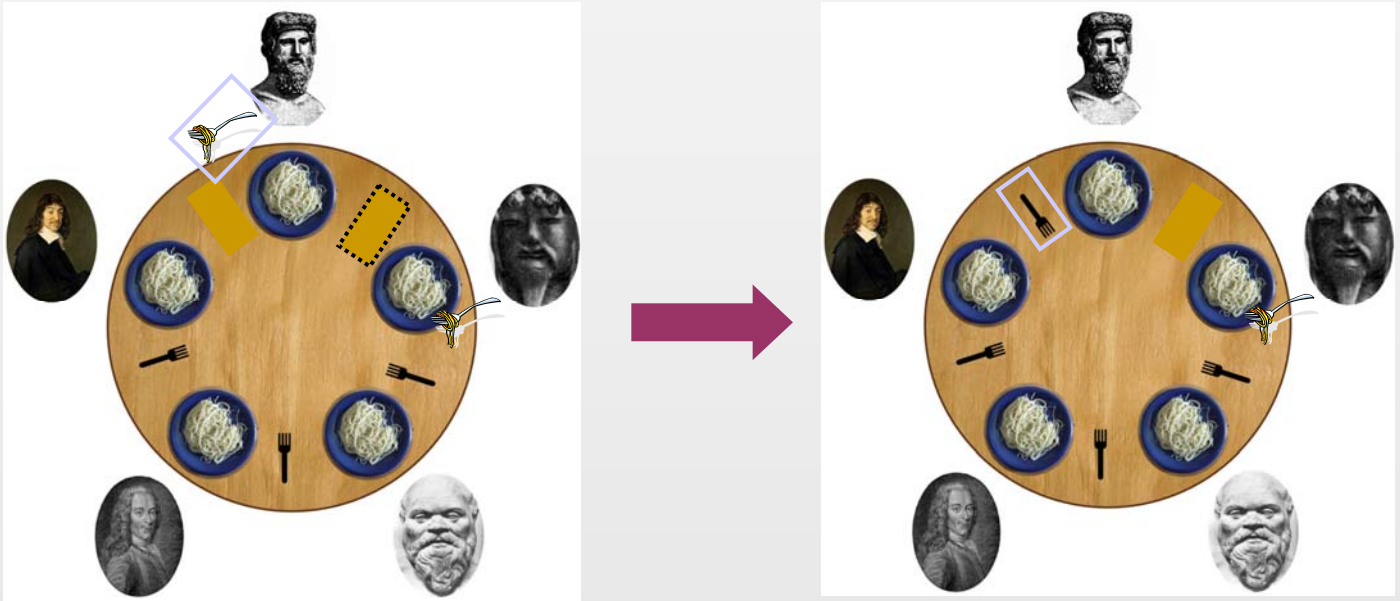
食事する哲学者の問題

- 制御せずに放って置くと、全員が同じ方の手(たとえば右手)でフォークを持った状態から進まなくなる
  - 左側にフォークが置かれるのを待ってしまうため

→ デッドロック

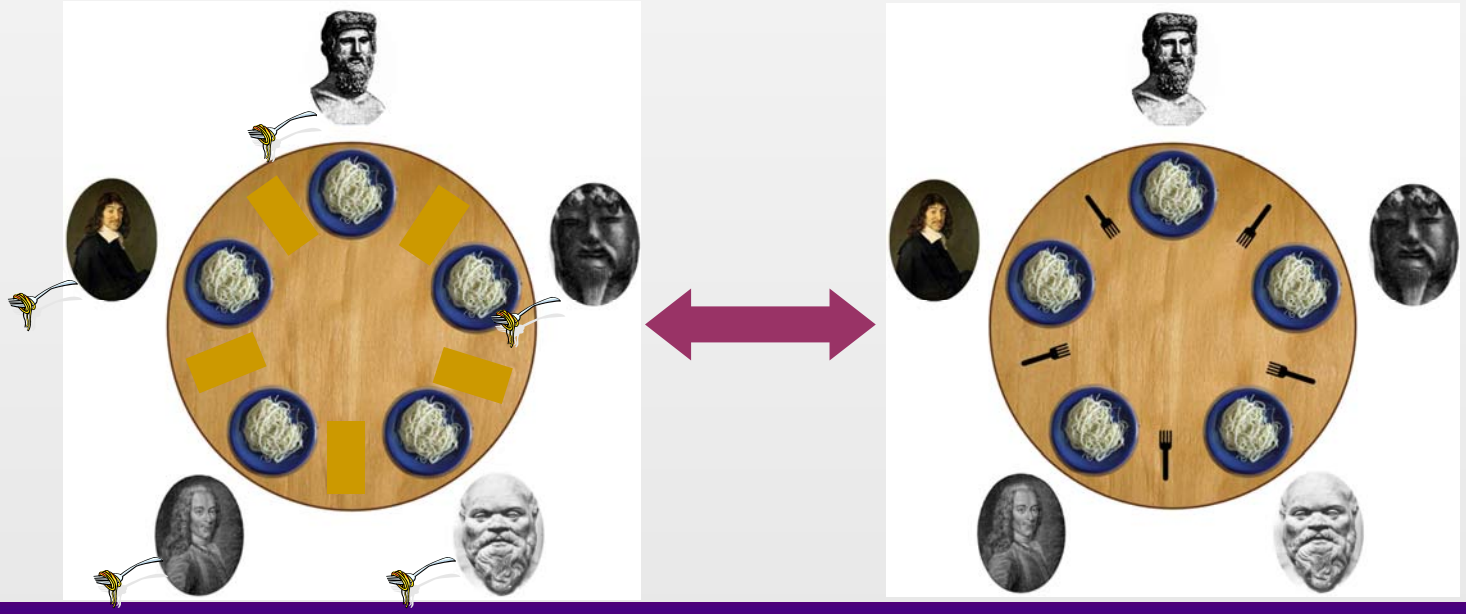
食事する哲学者の問題

- デッドロック回避策の例
  1. 最初に右側のフォークを取る
  2. 次に左側のフォークが既に取られていたならば、一旦右手のフォークを置く

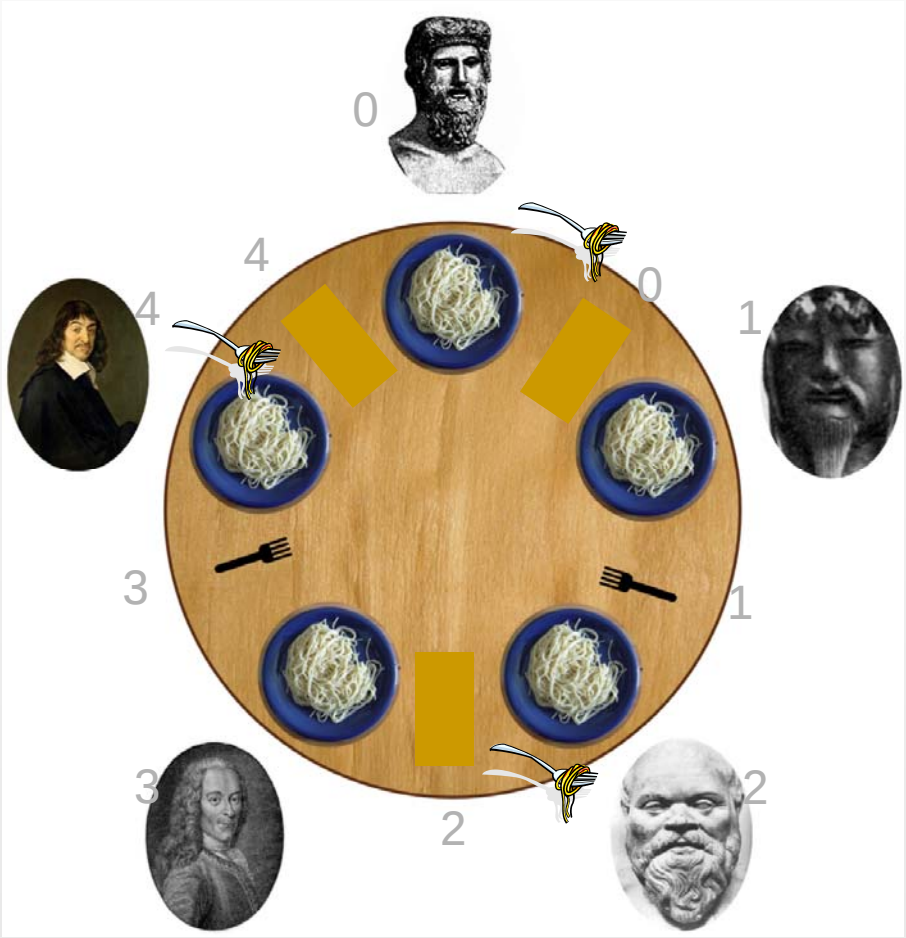


食事する哲学者の問題

- 問題点：次の1.と2.の繰り返しでライブロック発生
  1. 全哲学者が、一斉に右側のフォークを取り上げる
  2. 全哲学者が一斉に右手のフォークを置く
     - 全哲学者にとって、左側のフォークが取り上げられた状態であるため



食事する哲学者の問題



- デッドロック・ライブロック回避策の例
  - 奇数番号の哲学者は右側のフォークを先に取る
  - 偶数番号の哲学者は左側のフォークを先に取る

→ 4人(偶数)の場合は？

設計モデル例：食事をする哲学者のモデル化

並行同期システムでの設計の場合

- 順番に食事
- 偶数人の場合⇒奇数と偶数の人が交互に食事

同期遷移モデルの特徴

長所

- 同時に起こる動作を直接的に表現できる
- 通信がどのように起こるかを気にしなくてもよい
- モデルが単純になる

短所

- デッドロックや例外が発生しやすい

演習2：分散レコーダシステム

（演習2－A）DVD－HDレコーダ間のコピーを非同期通信、同期通信でモデル化せよ

（演習2－B）各設計モデルが異なることを確認し、それぞれの設計モデルの着眼点、特徴をまとめよ

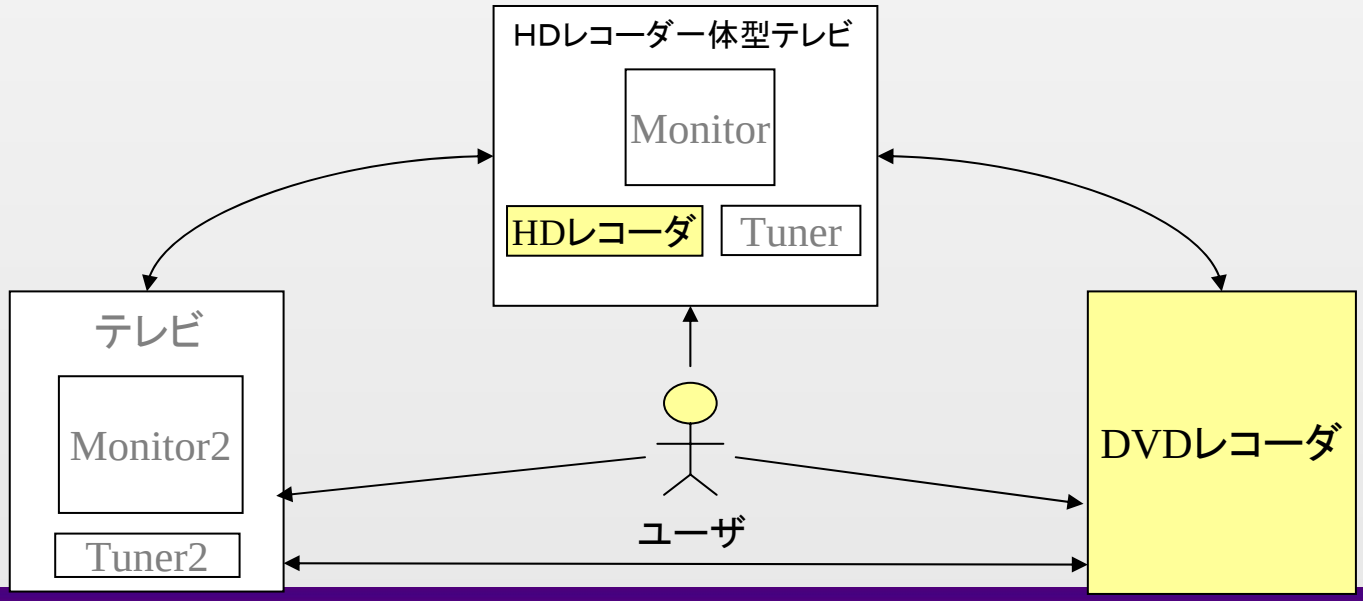
（演習2－C）シナリオ・テストに基づき、それぞれのモデルの正当性を確認せよ

分散レコーダシステム：要求仕様

- 一台のHDレコーダと一台のDVDレコーダはネットワークで接続されている。
- HDレコーダに格納されたコンテンツをDVDレコーダでDVDにコピーする。
- HDレコーダとDVDレコーダはネットワークで接続されており、利用にあたっては、事前にそのリソースを確保する必要がある。また利用後は他のネットワーク接続機器から利用できるように、リソースを開放する必要がある。リソースの確保、開放はローカルまたはリモートで行なわれる。
- 主成功系列は以下とする：
  (1) DVDレコーダ、HDレコーダの電源を投入する。
  (2) ユーザがDVDレコーダのコピーボタンを押下する。
  (3) DVDレコーダはコピー元のHDレコーダのコンテンツの再生を開始する。
  DVDレコーダはこの内容をDVDにコピーする。
  (4) HDレコーダの再生が終了し、DVDへのコンテンツのコピーが終了する。

分散レコーダシステム：全体システム構成

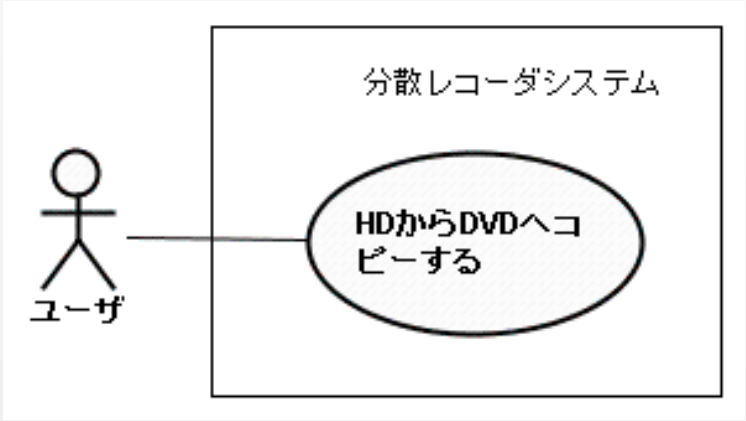
- ユーザ
  - DVDレコーダを操作して、HDレコーダのコンテンツをDVDにコピーするための指令をDVDレコーダに与える。
- DVDレコーダ
  - ユーザの操作に従って、HDレコーダのコンテンツを自身のDVDにコピーする。
- HDレコーダ
  - DVDレコーダからの指令に従って、HDのコンテンツを再生する。



分散レコーダシステム：検査仕様

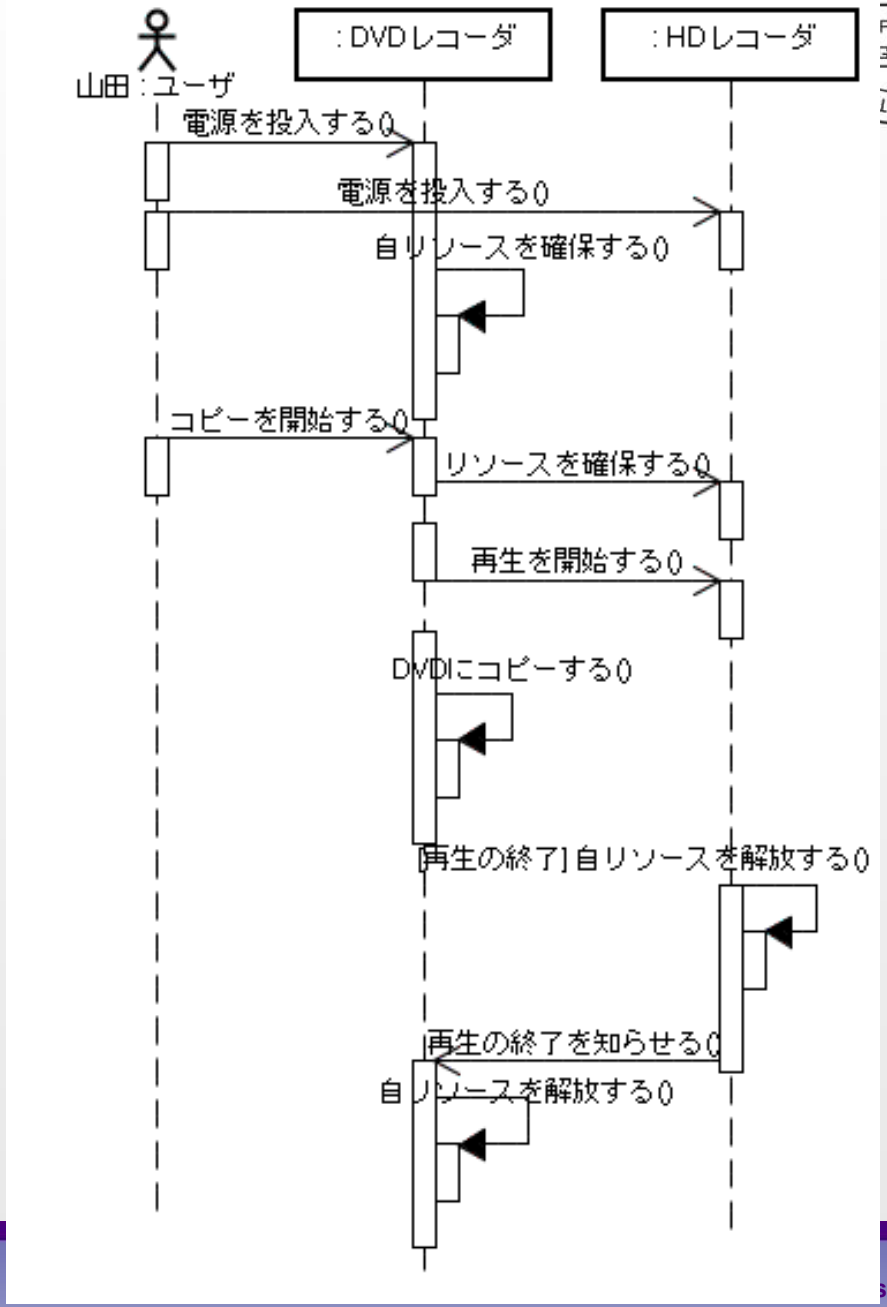
- 前提条件
  - DVDレコーダとHDレコーダのそれぞれのデバイスを利用してコピー操作を行う
  - コピー操作を行う場合、事前条件として、DVDレコーダ、HDレコーダのリソースが空いている
- 検査項目
  - コピー操作を行った場合，一連の流れの動作さが要求仕様どおりに動く
  - コピー終了後に再度同じ操作(コピー)を行っても，一連の流れが要求仕様どおりに動く

分散レコーダシステム：ユースケース

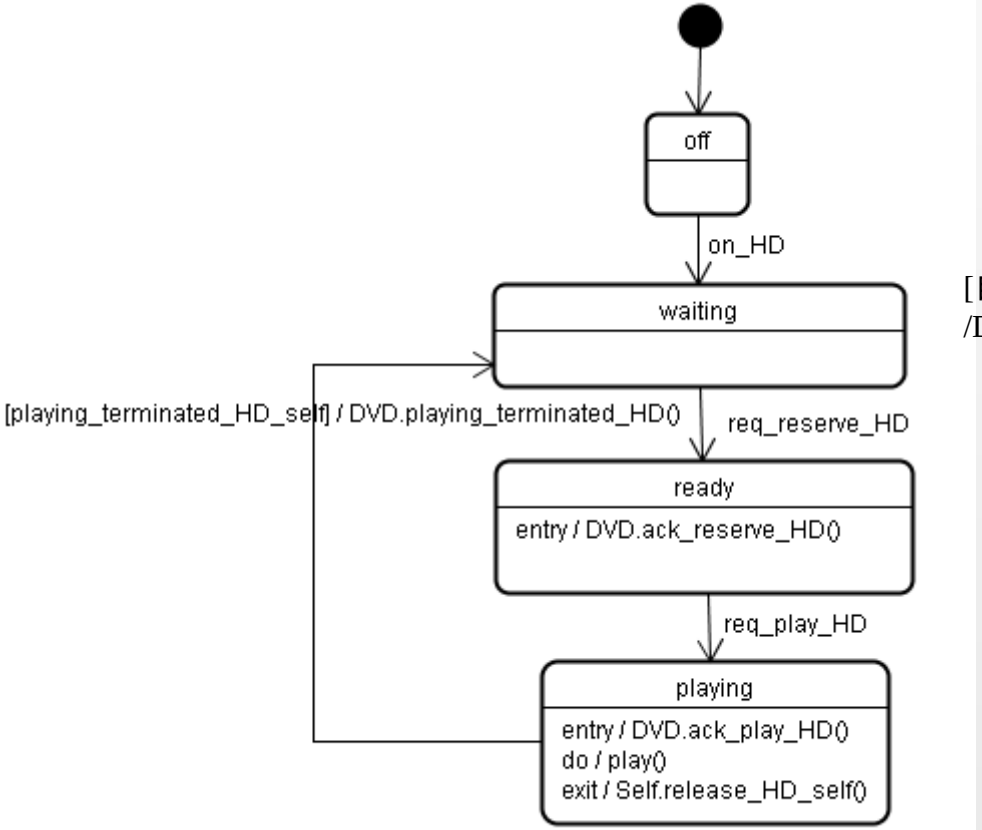


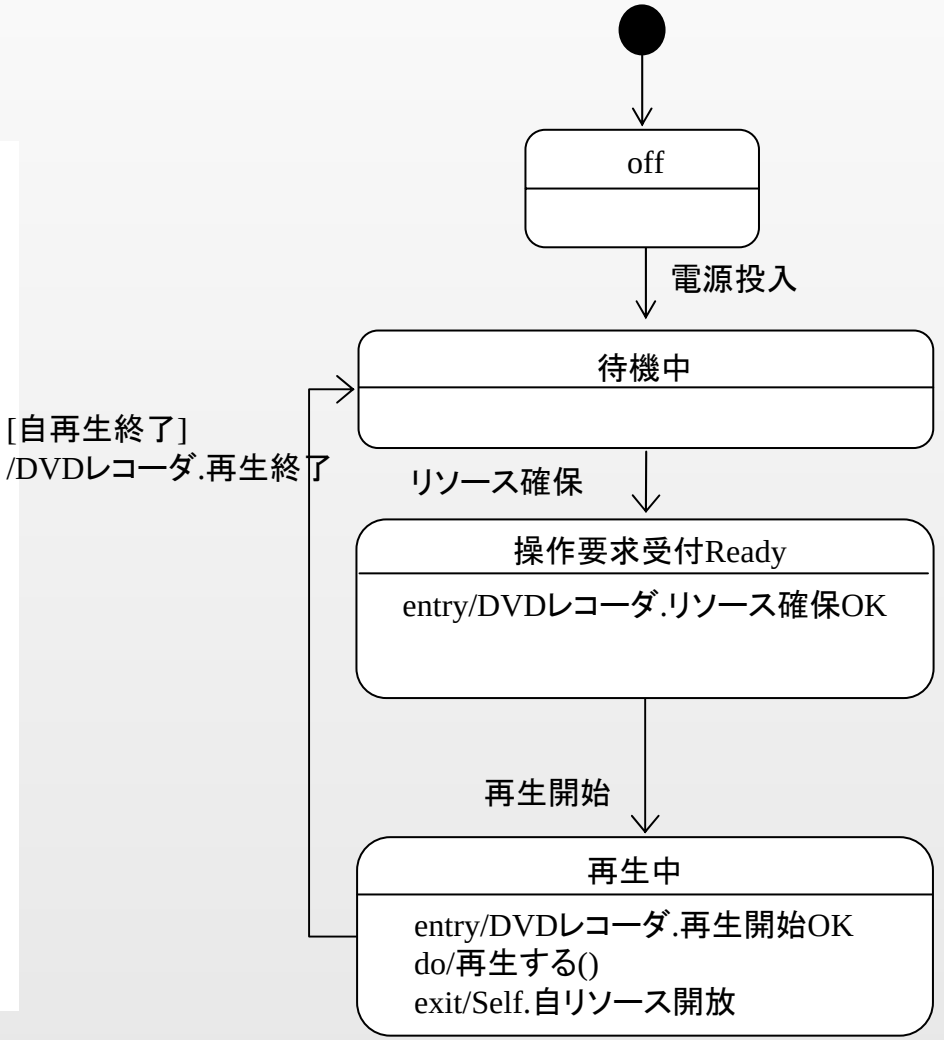
- ユースケース名: HDからDVDへコピーする
- 起動アクタ: ユーザ
- 事前条件: DVDレコーダ、HDレコーダの電源が落ちている。
- 事後条件: HDレコーダのコンテンツがDVDにコピーされている。
- 基本系列
  - 1. ユーザはDVDレコーダ、HDレコーダの電源を投入する
  - 2. ユーザがDVDレコーダにコピー処理を要求する
  - 3. DVDレコーダはHDレコーダのコンテンツ再生を開始し、その内容をDVDにコピーする
  - 4. HDレコーダの再生が終了し、DVDへのコンテンツのコピーが終了する

シーケンス図
シナリオ:コピーする

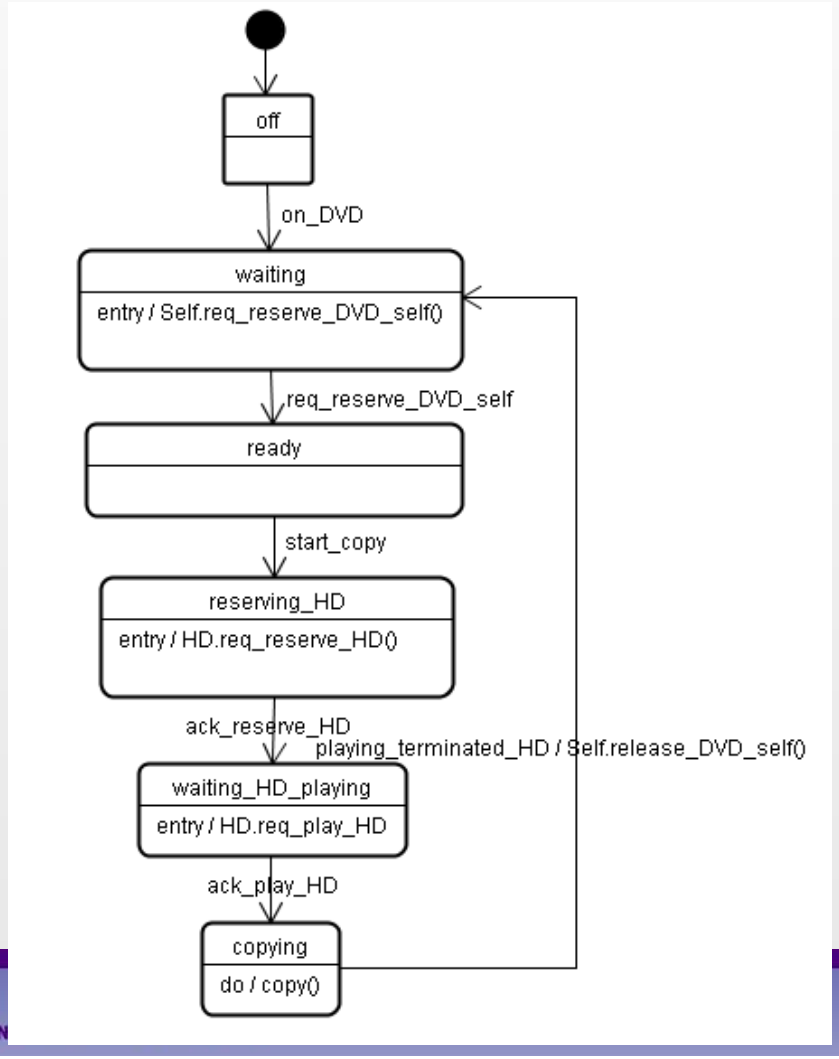


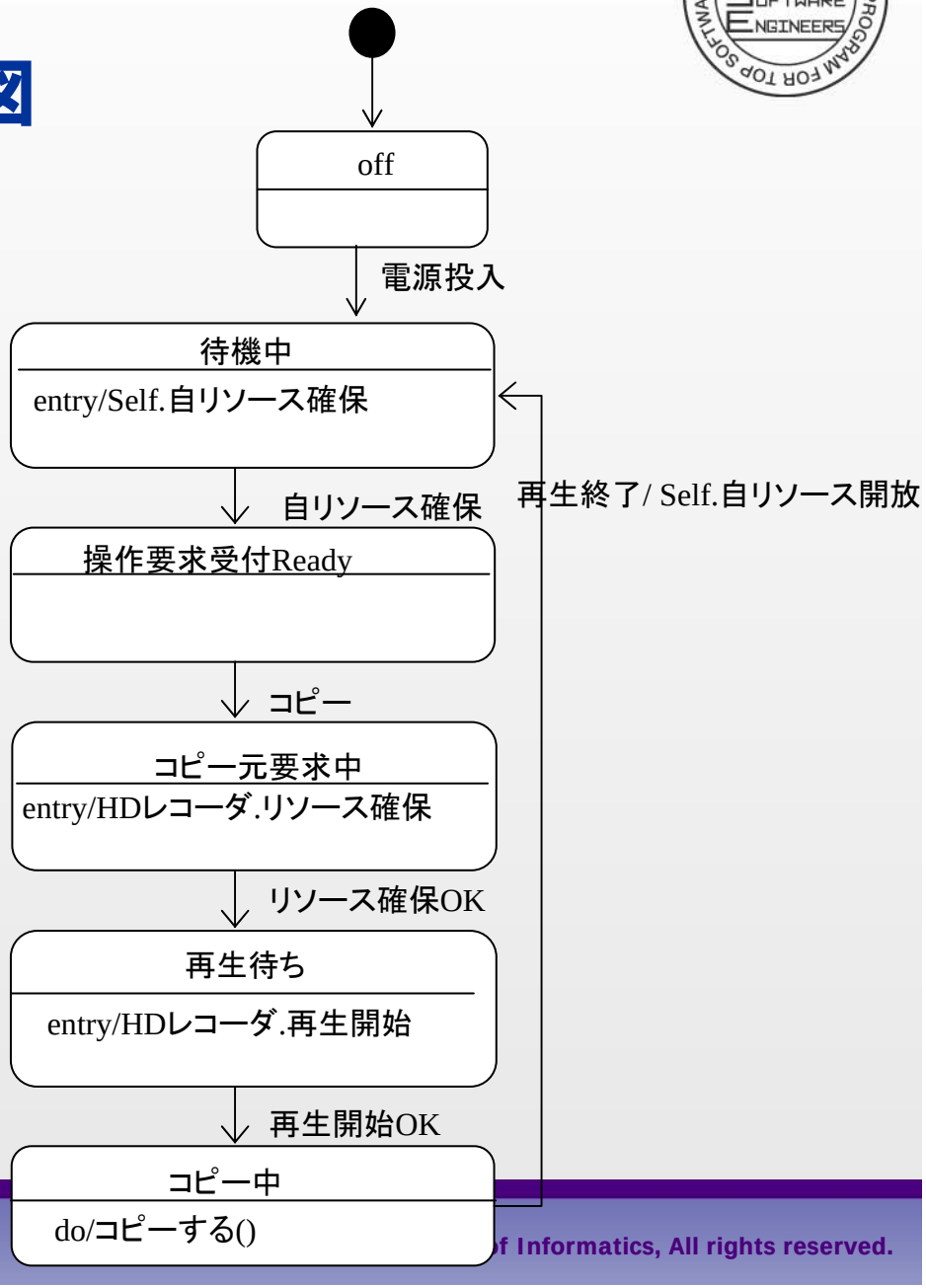
HDレコーダ：状態マシン図 ―非同期通信の場合―





DVDレコーダ：状態マシン図 ―非同期通信の場合―





同期通信・非同期通信のモデルの違い

【非同期通信】
長所
- 例外に対する詳細設計が可能
- 並列処理をモデル化しやすい

短所
- 相手の状態を仮定してモデルを作りづらい
- 例外への対応が難しい
  - 例外のバリエーションが多くなる

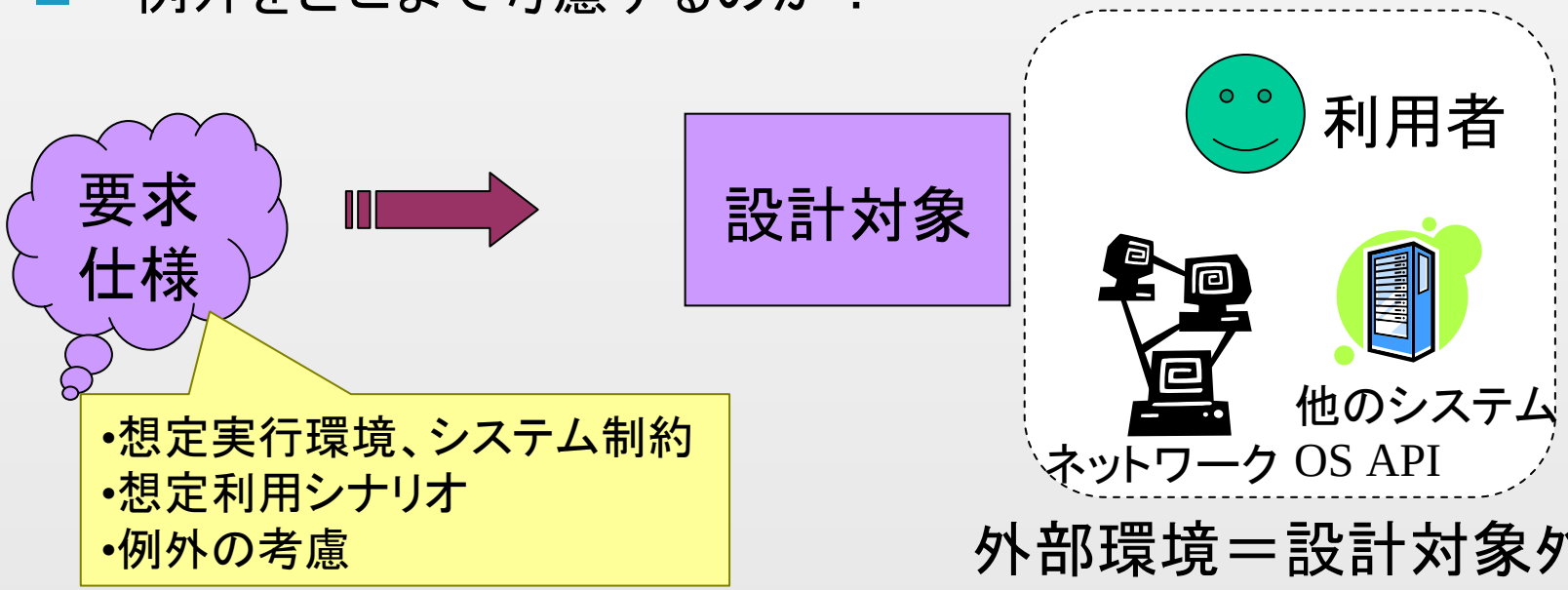
【同期通信】
長所
- 相手の状態を仮定したモデルが作れる
  - 通信が出来た時点で相手が処理可能になっていることが仮定できる
- 例外や並列処理をあまり考慮しない段階でロジックが検査可能

短所
- 並列処理が素直にモデル化できない
  - 通信で必ずブロックするので、相手が何かをしている間に特定の処理をするということを直接表現できない
  - タイムアウトを多用することで、レスポンスが遅い設計になりがち

外部環境の考慮した設計

なぜ外部環境を考慮する必要があるのか？

1. 閉じた環境で検証する必要があるため
2. 想定した環境により設計すべきモデルが異なるため
   - 環境はシステムと同期して動作するのか、非同期なのか？
   - 例外をどこまで考慮するのか？



外部環境＝設計対象外

演習３：外部環境を考慮した設計

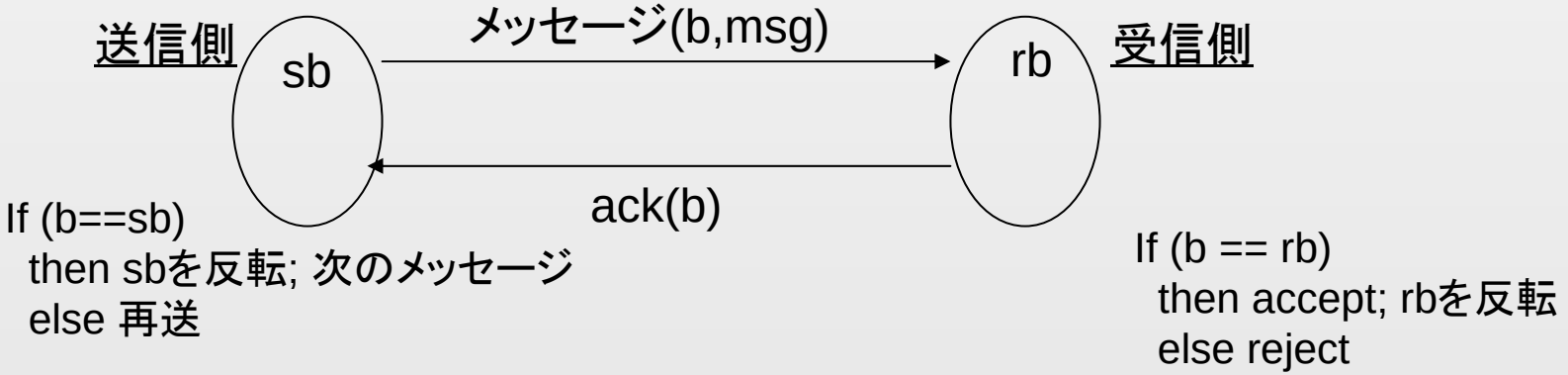
（演習３－Ａ）ネットワーク環境を考慮して、プロトコルを設計せよ

【ネットワーク環境例】

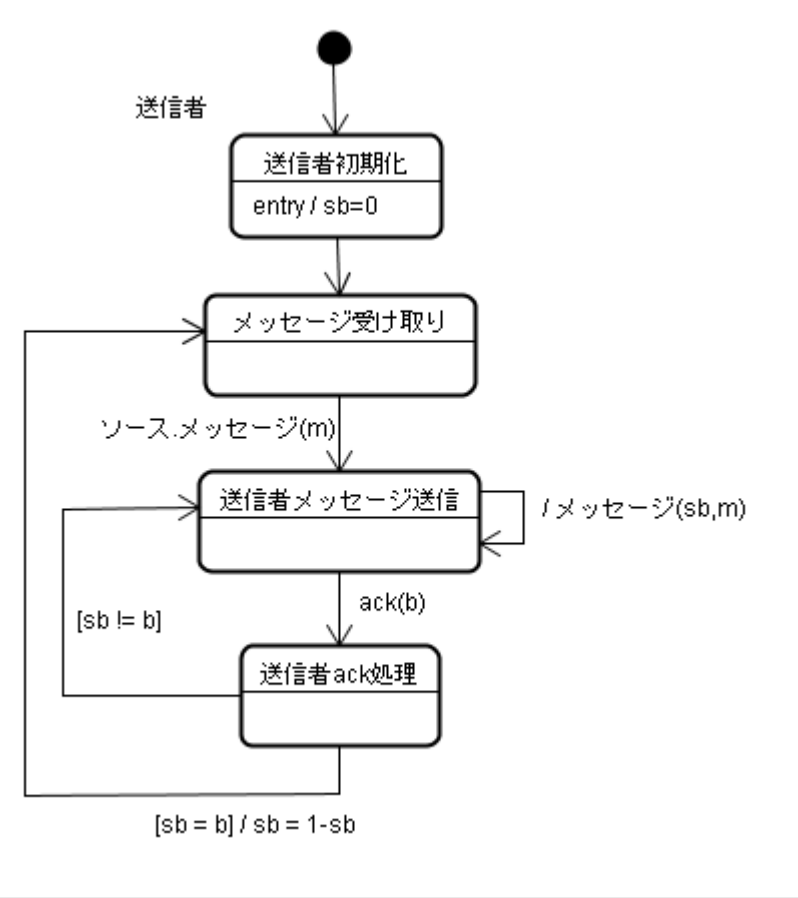
- 信頼できるネットワーク：常に正しく送信可能
- 信頼できないネットワーク
  - データが変化するネットワーク
  - 通信が喪失するネットワーク
  - 通信の順番が入れ替わるネットワーク

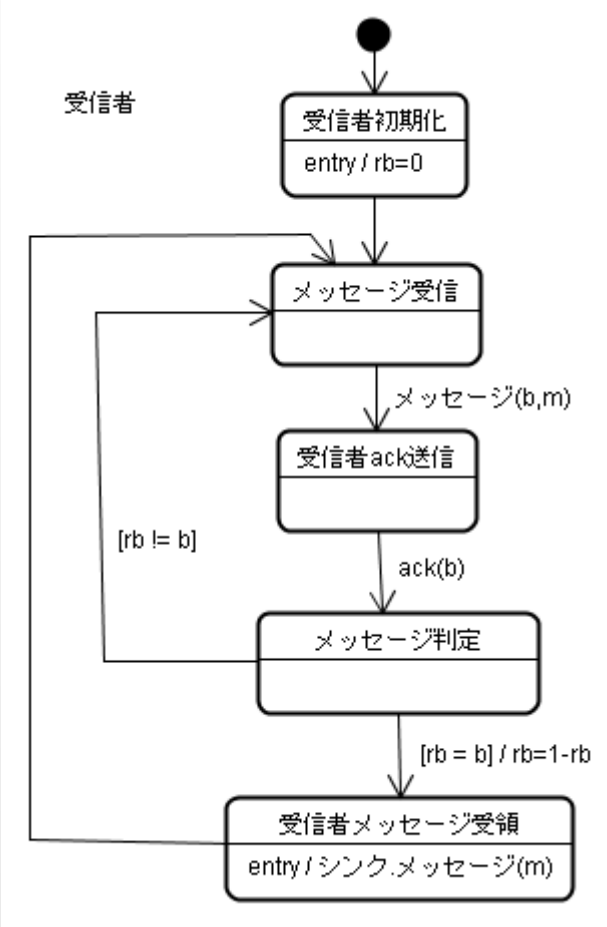
例題：Alternating Bit Protocol（ABP）

- 信頼できないネットワークを介して、正しくメッセージの転送を行うためのプロトコル設計例。
  - 送信側と受信側の双方が0と1の値を持つ変数（ヘッダ）をもっており、それとは異なるビットを持ったメッセージを受け取った場合、再送を促す。
- 片方向通信



ABPの設計モデル





演習３：外部環境を考慮した設計

（演習３－Ｂ）この設計はどこまでの外部環境を想定しているのかを整理せよ

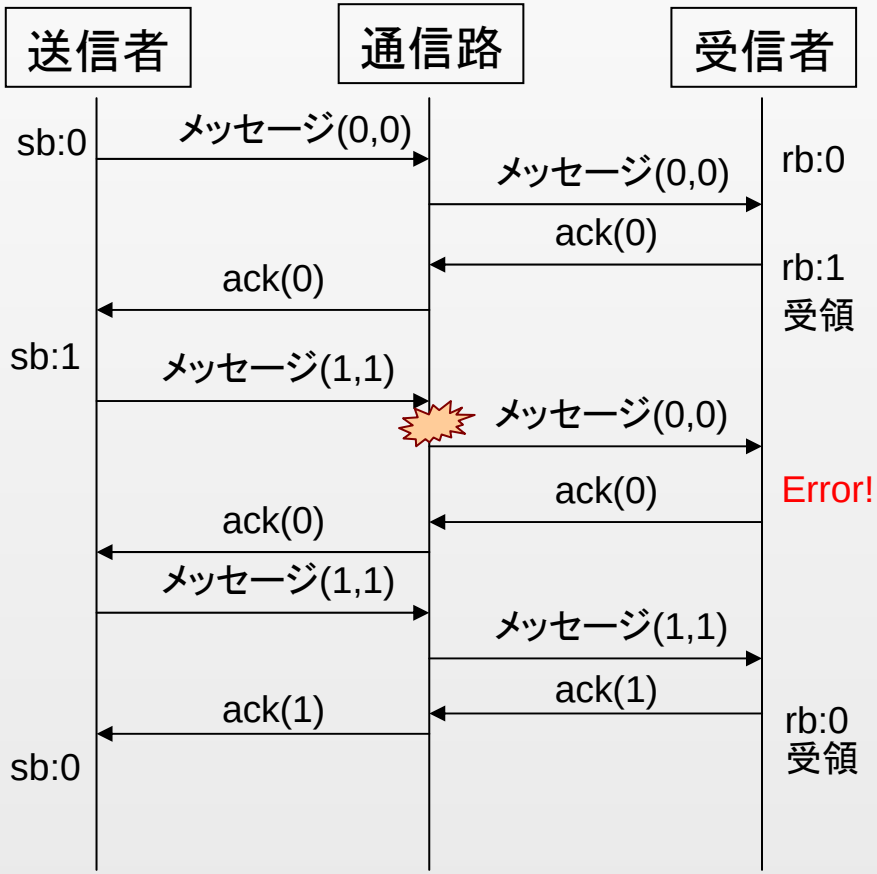
- 外部環境として何があるのか？
- 例外として何が考慮されているのか？

（演習３－Ｃ）他の想定環境ではどう設計が変わってくるのかを整理せよ

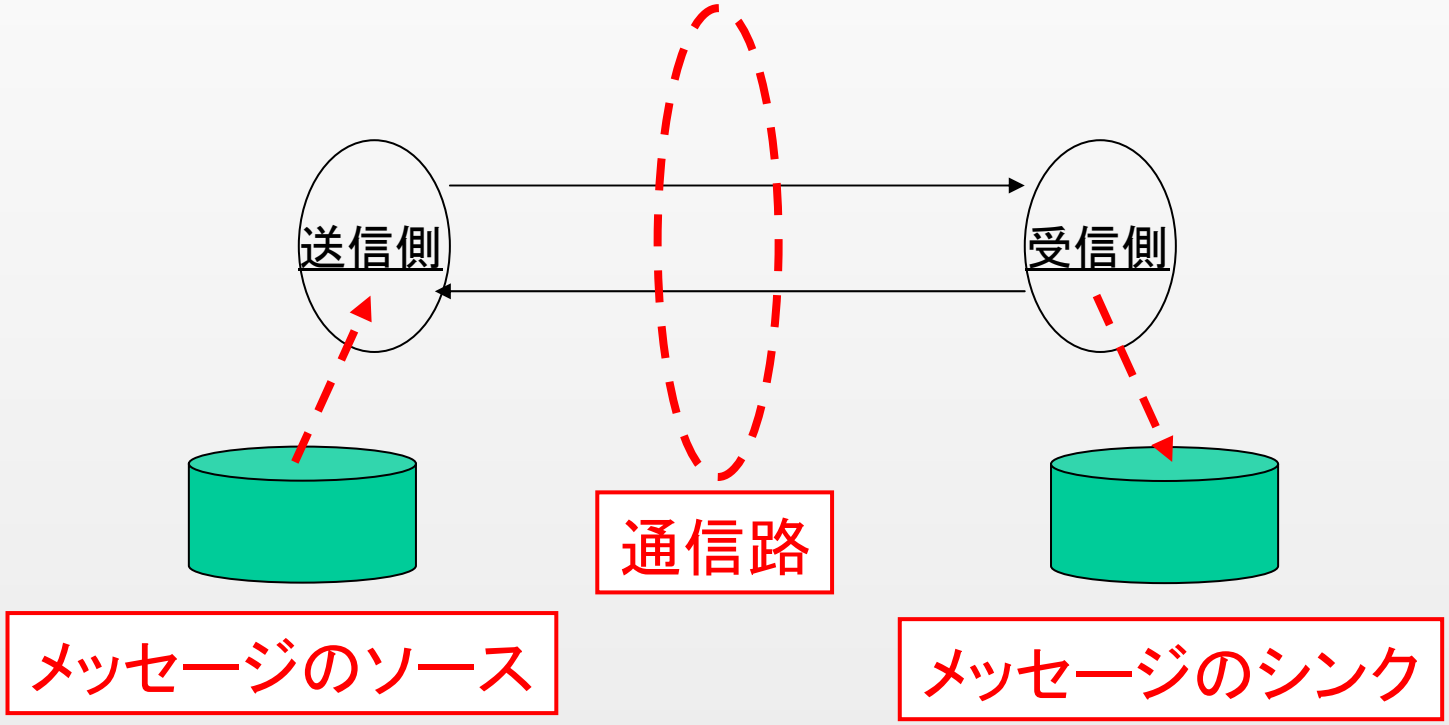
- データが誤るネットワーク
- 通信が消失するネットワーク
- 通信の順番が入れ替わるネットワーク

（演習３－Ｄ）外部環境をモデル化し、それに合わせたプロトコルを設計せよ

メッセージ通信例

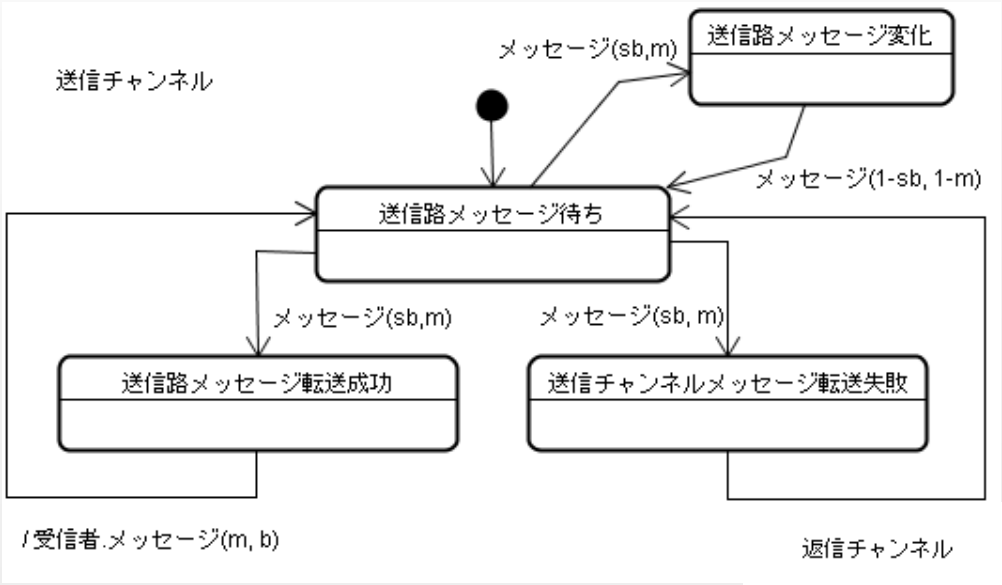


解答：外部環境



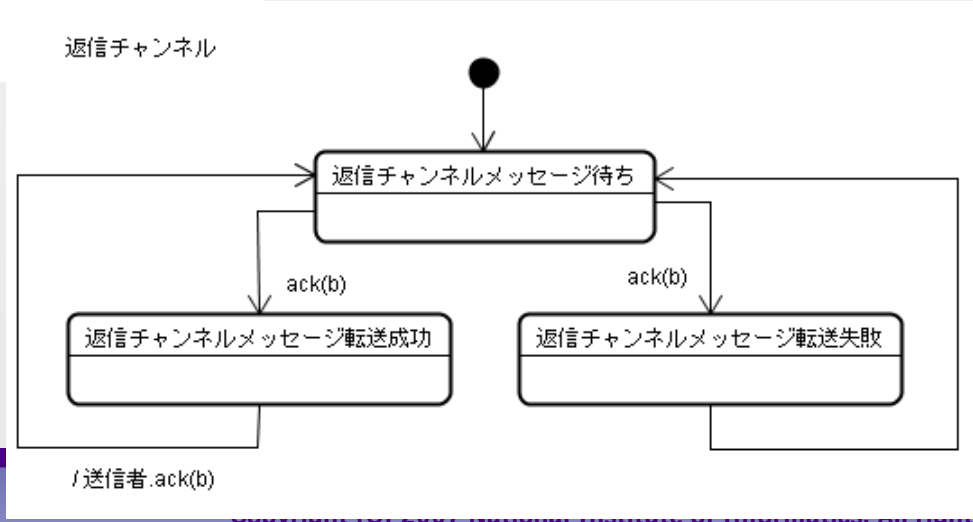
通信路のそれぞれの通信経路について環境の振る舞いを考える必要がある

通信路のモデル例



例外１）データ誤り
- メッセージ本体のみ誤り
- 全て誤り

例外２）データ消失
- 全て消失
- メッセージのみ消失



まとめ

- 基礎編で何を学んだか?
  - 分散システムの難しさ
    - 網羅性の検証の難しさ、組み合わせ爆発
    - 振る舞いの複雑さ
  - 検証プロセスの概要
- 応用編では何を学ぶか?
  - システムの制約を考慮した設計と検証:同期・非同期遷移、同期・非同期通信
  - 環境のモデリング:切断、順序の入れ替わり、メッセージ変更
- 並行システムのモデル化法
  - 同期・非同期遷移、非同期・同期通信の違い、特徴
  - システム制約・条件を考慮した設計モデルの構築
- 外部環境を考慮した設計モデルの構築

宿題・レポート

- 3つの演習をまとめる

（演習4）各自、同期、非同期、外部環境を考えた応用問題を設定せよ

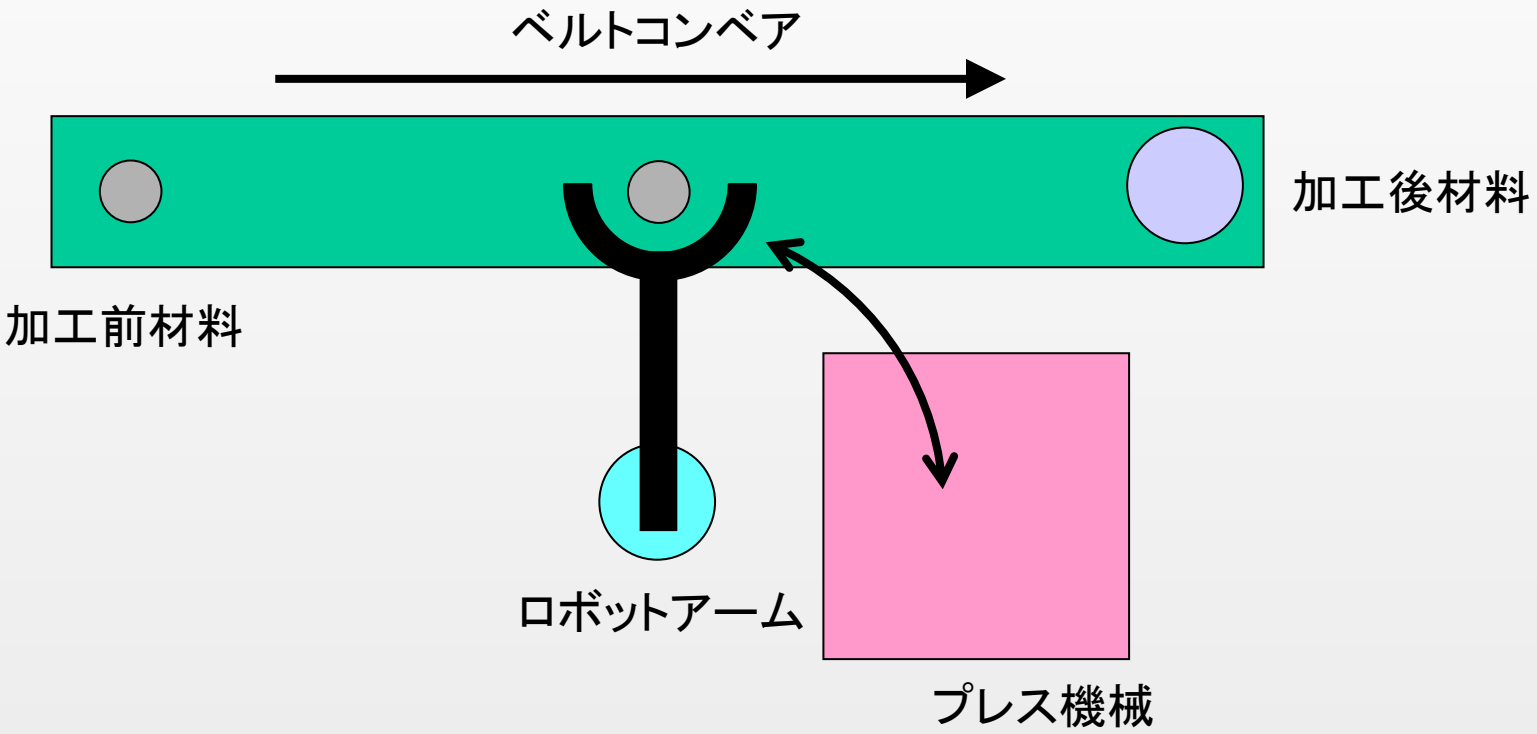
  - システムの制約・条件、検証項目（考慮すべき例外）を各自設定
  - 次回以降、各自の例題を使って演習

- 期限：次回の講義（9月14日）前まで
- 提出先：http://netcommons.topse.jp/
  - H21設計モデル検証（応用編）

応用問題の設定

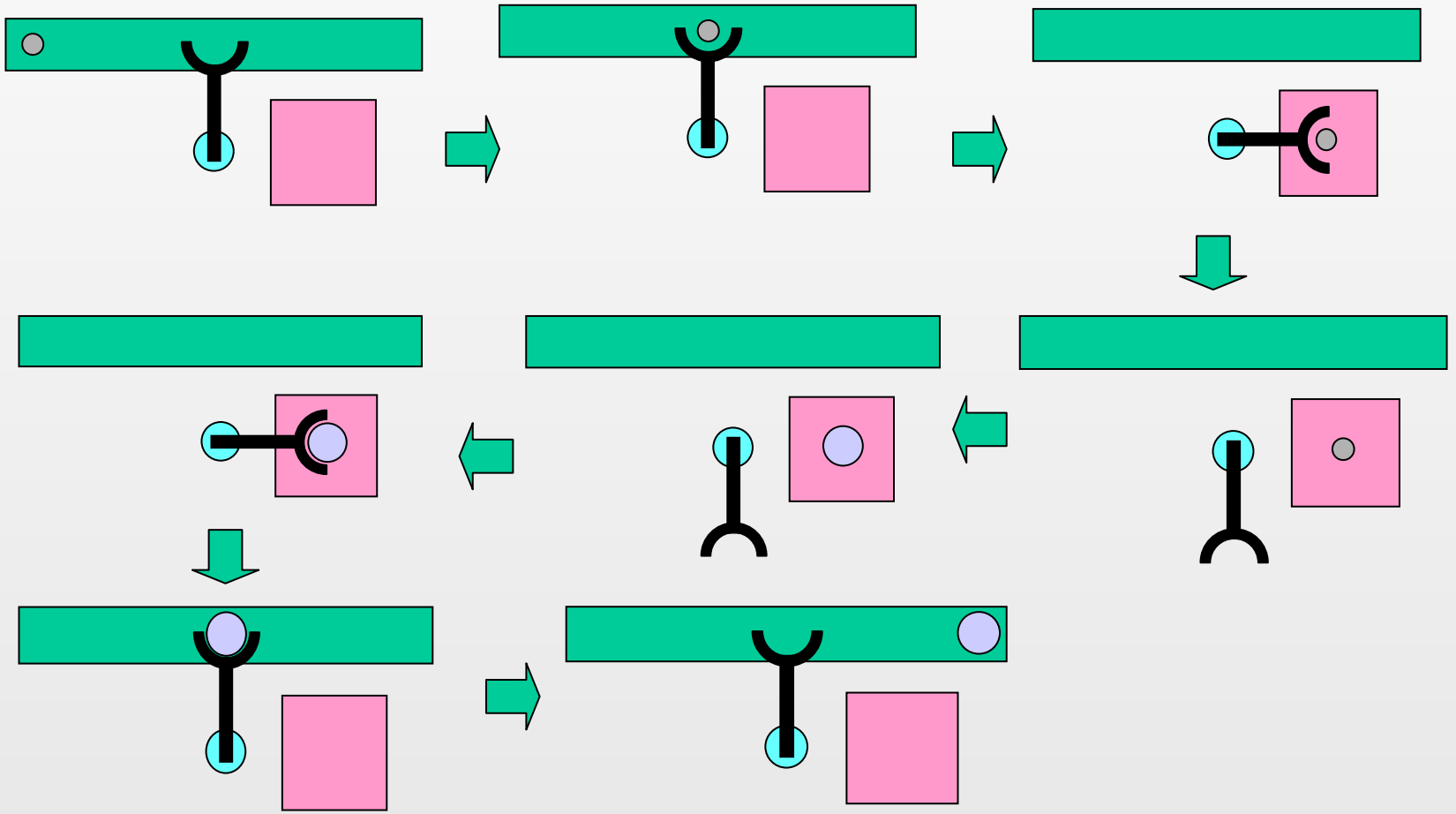
- 各自の興味のある例題を設定ください。
  - 身近な問題や業務・研究に関係のあるものをなるべく選んでください
- 問題には、下記が含まれます。
  - 要求仕様：何をしたいのかを自然言語＋ユースケースで明記
  - システムの前提、仮定：自然言語＋UML図
  - 設計する範囲と例外の範囲：自然言語＋ユースケース記述（例外シナリオ）
- 特に同期遷移や同期通信でモデル化したら良いと思われる例を挙げてください
- 一つのシステムについて、同期、非同期で考えた場合どうなるかという問題でも構いません
  - その場合、同期、非同期で、それぞれどのような前提・仮定を想定しているかを明確にしてください。
- 出来れば、ラフな設計モデルも書いてきてください

応用問題例：生産ライン制御



- ベルト上には、左、中心、右に各1つずつ、計3つまで材料を配置可能
- ロボットアームとプレス機械は、ベルトの真ん中に材料が到着次第、直ちに加工しベルトに戻す
- ベルトの左方からは、ベルトの左部が空いていれば、随時材料が供給され、右方では材料が到着次第次工程に移動される

動作例：生産ライン制御



要求仕様例

- 加工前材料が供給されれば、そのうち材料が加工される
- 材料が加工されれば、加工後材料がベルト右方に到着する
- アームが材料保持中、または材料加工中は、材料はベルトの中心部にはない
- 材料加工中は、ロボットアームはプレス機械上にはない